White Rose Research Online

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk
Universities of Leeds, Sheffield and York

Deposited via The University of Sheffield.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/191518/

Version: Accepted Version

Article:
Popescu, A. and Roşu, G. (2015) Term-generic logic. Theoretical Computer Science, 577. pp. 1-24. ISSN: 0304-3975

https://doi.org/10.1016/j.tcs.2015.01.047

Article available under the terms of the CC-BY-NC-ND licence (https://creativecommons.org/licenses/by-nc-nd/4.0/).

Reuse
This article is distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs (CC BY-NC-ND) licence. This licence only allows you to download this work and share it with others as long as you credit the authors, but you can't change the article in any way or use it commercially. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Term-Generic Logic

Andrei Popescu[a,b], Grigore Roşu[c,d]

[a] *Chair of Logic and Verification, Technische Universität München*
[b] *Institute of Mathematics "Simion Stoilow" of the Romanian Academy*
[c] *Department of Computer Science, University of Illinois at Urbana-Champaign*
[d] *University Alexandru Ioan Cuza, Iaşi, Romania*

Abstract

We introduce *term-generic logic (TGL)*, a first-order logic parameterized with terms defined axiomatically (rather than constructively), by requiring terms to only provide *free variable* and *substitution* operators satisfying some reasonable axioms. TGL has a notion of model that generalizes both first-order models and Henkin models of the λ-calculus. The abstract notions of term syntax and model are shown to be sufficient for obtaining the completeness theorem of a Gentzen system generalizing that of first-order logic. Various systems featuring bindings and contextual reasoning, ranging from pure type systems to the π-calculus, are captured as theories inside TGL. For two particular, but rather typical instances—untyped λ-calculus and System F—the general-purpose TGL models are shown to be equivalent with standard ad hoc models.

Keywords: term-generic logic, substitution, λ-calculus, π-calculus, semantics

1. Introduction

First-order logic (FOL) does not allow variables to be bound in terms (but only in formulas, via quantifiers), thus providing a straightforward notion of substitution in terms. On the other hand, most calculi and type systems used in programming languages are crucially based on the notion of binding of variables *in terms*: terms "export" only a subset of their variables, the free ones, that can be substituted. Because of their complex formulation for terms, these calculi cannot be naturally captured as FOL theories. Consequently, they need to define their own models and deduction rules, and to state their own theorems of completeness, not always easy to prove. In other words, they are presented as entirely new logics, as opposed to theories in an existing logic, thus incurring all the drawbacks (and boredom) of repeating definitions and proofs following generic, well-understood patterns, but facing new details.

Email addresses: uuomul@yahoo.com (Andrei Popescu), grosu@illinois.edu (Grigore Roşu)

In this paper we define *term-generic first-order logic*, or simply *term-generic logic (TGL)*, as a many-sorted first-order logic parameterized by any terms that come with abstract notions of *free variable* and *substitution*. More precisely, in TGL terms are elements in a generic set *Term* (including a subset *Var* whose elements are called variables) that comes with functions $FV : Term \to \mathcal{P}(Var)$ and $Subst : Term \times Term^{Var} \to Term$ satisfying some axioms, including compositionality for substitution. Terms are classified according to sorts and substitution is compatible with sorting. TGL models provide interpretations of terms, subject to axioms requiring smooth interaction with the syntactic operators (Section 2).

A main contribution of this paper is demonstrating that the above axiomatization is sufficient to develop the fundamental theory of first order-logic up to the completeness theorem. TGL admits a complete Gentzen deduction system, syntactically similar to that of FOL; its proof of completeness modifies the classic proof of completeness for FOL to use the generic notions of term, free variables and substitution (Section 3).

TGL can be instantiated to different kinds of terms, such as standard FOL terms or different categories of (typed or untyped) λ-terms. When instantiated to standard FOL terms, TGL becomes, as expected, precisely FOL. However, when instantiated to more complex terms, e.g., the terms of λ-calculus, TGL becomes a logic where a particular calculus is a particular theory.

We give several examples of calculi and type systems that can be described as TGL theories, including the λ-calculus and the π-calculus (Section 4). E.g., the β rule of λ-calculus is represented by the TGL axiom-schema in a language having the terms of λ-calculus and a binary relation symbol $\rightsquigarrow$:

$$(\lambda x.X)\; x \rightsquigarrow X \qquad (*)$$

assumed parameterized by the term X and having all its free variables quantified at the top by an implicit $\forall$. The variable x is bound in the left X by the term operator λ; the occurrences of x outside of $\lambda x.X$, that is, the second listed x and any occurrence of x in the right X, are bound by the top $\forall$ quantifier.

The λ-calculus mechanisms are managed in TGL by FOL-like reasoning mechanisms. Consider the reduction

$$(\lambda x.\, x\, x)\, (\lambda x.\, x) \rightsquigarrow (\lambda x.\, x)\, (\lambda x.\, x)$$

It is deduced in TGL from (*) taking X to be $x\, x$ and instantiating the $\forall$-bound x with $\lambda x.\, x$. As customary in first-order logic, $\forall$-instantiation takes place by substitution, only now one employs the λ-calculus syntax for terms. After instantiation, $\lambda x.X$ stays the same (since $(\lambda x.x\, x)[(\lambda x.\, x)/x] = \lambda x.x\, x$), the second listed x becomes $\lambda x.x$ (since $x[(\lambda x.\, x)/x] = \lambda x.x$), and the right X becomes $(\lambda x.\, x)\, (\lambda x.\, x)$ (since $(x\, x)[(\lambda x.\, x)/x] = (\lambda x.\, x)\, (\lambda x.\, x)$). In general, instantiating in (*) the $\forall$-bound x with arbitrary terms Y, we obtain the more familiar schema used in λ-calculus, $(\lambda x.X)\; Y \rightsquigarrow X[Y/x]$.

Type systems can also be represented in TGL. For example, the following rule for typing abstractions in typed λ-calculi,

$$\frac{\Gamma, x : T \rhd X : T'}{\Gamma \rhd \lambda x{:}T.\, X : T \to T'}[x \text{ fresh for } \Gamma]$$

is represented by the following TGL axiom-schema written in a language with 2 sorts, one for data and one for types, and a binary relation symbol *tpOf* of arity $data \times type$:

$$(\forall x.\, tpOf(x,t) \Rightarrow tpOf(X,t')) \Rightarrow tpOf(\lambda x{:}t.\, X,\, t \to t')$$

In this schema, x and t, t' are data and type variables, respectively, X is an arbitrary data term (a parameter of the schema), and $\Rightarrow$ is logical implication. The term X may contain the free variable x, which is bound by TGL's $\forall$ in the left of the outermost implication, and by the term syntax's λ in the right.

As illustrated by the above examples, the TGL descriptions are more compact and more high-level than the original presentations of the rules: for β there is no explicit substitution and for typing there is no side-condition or explicit context—all these details are managed by the underlying mechanism of TGL.

A description of a calculus as a TGL theory automatically explains a notion of model for that calculus: the TGL models satisfying the theory. Models for λ-calculi have received much attention in the literature, as witnessed in the monographs [7, 25, 29]. Apart from their intrinsic interest as the dual of syntax, models are often used to get a better insight into a calculus or a type system, which helps developing its metatheory. For example, Reynold's abstraction theorem (later called parametricity) [43], as well as Wadler's free theorems [52], make essential use of models. More generally, the method of logical relations [46] can be developed systematically using models [29].

A natural question is how insightful are these general-purpose TGL models for the described calculi. Due to the large palette of systems describable in TGL and the bewildering variety of methods and models developed for existing systems, a complete answer to this question cannot be attempted. However, we can identify some characteristics of the TGL models: they offer a loose, set-theoretic semantics to a calculus. We look at two well-known particular cases, considering standard ad hoc models for the λ-calculus and System F falling in this category (Section 5). We show that, in these cases, the free TGL models are equivalent to the ad hoc models.

While it captures various systems with bindings as particular theories, TGL does not address the mechanical representations of the syntax of these systems, suitable for reasoning in a theorem prover. Indeed, TGL does not indicate means to represent syntax with bindings, but is parameterized by such a syntax. Nor does it address the mechanical representation of its axiom-schemas. On the other hand, higher order abstract syntax (HOAS) is a methodology specialized in precisely these two aspects: syntax with bindings and schematic judgments. We discuss the relationship between TGL and HOAS and the potential benefit of combining the two. Most proofs are delegated to the appendix.

This paper extends the WADT'08 conference paper [41] with the following: an analysis of the use of the generic-syntax axioms in the proof of completeness (Section 3); a TGL representation for the π-calculus (Subsection 4.3); a comparison between TGL models and ad hoc models (Section 5); proofs of the facts stated in the paper. One aspect not included in this paper, but discussed

in the technical report [40], is a methodology for establishing adequacy of TGL specifications, based on a fragment of the logic called HORN^2 (Horn squared).

Conventions and notations. If $i, j \in \mathbb{N}$, then $\overline{i,j}$ denotes the set of natural numbers between i and j, inclusively. Overlined single symbols such as $\overline{T}$ indicate tuples $(T_1, \ldots, T_n)$. A^* is the set of finite words (sequences) over A. $\mathcal{P}_f(A)$ is the set of finite subsets of A. Given two sets A and B, both B^A and $A \to B$ denote the set of functions from A to B. "Function", "map" and "mapping" will be used interchangeably. 1_A is the identity function on A. If $f : A \to B$, $a \in A$ and $b \in B$, then $f[a \leftarrow b]$ is the function that maps all $a' \neq a$ to $f(a')$ and a to b. Given $f : A \to B$ and $A' \subseteq A$, $f\restriction_{A'}\colon A' \to B$ is the restriction of f to A'. Given a property P and a function f, we write $\{f(a) \mid P(a)\}$ for the set of all items of the form $f(a)$ where a is in the domain of f and $P(a)$ holds. We sometimes inline function definitions, e.g., writing $a \mapsto a + 5$ for the function that maps each a to $a + 5$.

2. Term-Generic First-Order Logic

We introduce a generic notion of first-order term, axiomatized by means of free variables and substitution, which we call a *term syntax*. Then we proceed with a development of many-sorted first-order logic parameterized by a term syntax, proving basic sanity facts about substitution and model interpretation of terms and formulas.

2.1. Generic Term Syntax

Roughly, a term syntax is a collection of objects called terms endowed with free variables and a well-behaved mechanism for substitution.

Definition 2.1. *Let S be an (at most) countable set of sorts and Var a countably infinite set of variables. A* term syntax *over (S, Var) consists of:*
(a) A set $Term$ such that $Var \subseteq Term$, whose elements are called terms*;*
(b) A sort-assigning function $sort : Term \to S$;
(c) A mapping $FV : Term \to \mathcal{P}_f(Var)$; the elements of $FV(T)$ are called free variables*, or simply* variables*, of T;*
(d) A mapping $Subst : Term \times Term^{Var,sort} \to Term$, called substitution*, where $Term^{Var,sort}$ is the set of sort-preserving functions from Var to $Term$, $\{\theta : Var \to Term.\ \forall x \in Var.\ sort(\theta(x)) = sort(x)\}$.*

These are subject to the following requirements, where s ranges over sorts, x over variables, T, T' over terms, and θ, θ' over $Term^{Var,sort}$:
(0) $\{x \in Var.\ sort(x) = s\}$ is infinite and $sort(Subst(T, \theta)) = sort(T)$;
(1) $Subst(x, \theta) = \theta(x)$;
(2) $Subst(T, Var \hookrightarrow Term) = T$;
(3) If $\theta\restriction_{FV(T)} = \theta'\restriction_{FV(T)}$, then $Subst(T, \theta) = Subst(T, \theta')$;
(4) $Subst(Subst(T, \theta), \theta') = Subst(T, \theta\,;\theta')$, where the composition $\theta\,;\theta'$ is defined by $(\theta\,;\theta')(y) = Subst(\theta(y), \theta')$ for all $y \in Var$;
(5) $FV(x) = \{x\}$;
(6) $FV(Subst(T, \theta)) = \bigcup\{FV(\theta(x)).\ x \in FV(T)\}$.

Note that we assume the notion of term coming together with a notion of substitution which is *compositional* (condition (4) above). In general, for a syntax with bindings, compositionality of substitution naturally holds for α-equivalence classes—for raw terms, this property can be imposed only with some care in the choice of the fresh variables [47]. In the concrete instances of our logic to calculi with bindings, TGL terms will be α-equivalence classes of what are usually called (raw) "terms" in these calculi. Condition (0) is a well-sortedness requirement: there is an infinite supply of variables for each sort and substitution is sort-preserving. Conditions (1)-(6) are natural (and well-known) properties of substitution holding for virtually all notions of terms with static binding (modulo α-equivalence).

We fix a term syntax $(\mathit{Term}, \mathit{sort}, \mathit{FV}, \mathit{Subst})$. For distinct variables $x_1, \ldots, x_n$ and terms $T_1, \ldots, T_n$ such that $\mathit{sort}(T_i) = \mathit{sort}(x_i)$, we write $[T_1/x_1, \ldots, T_n/x_n]$ for the function $\mathit{Var} \to \mathit{Term}$ that maps x_i to T_i and all the other variables to themselves, and $T[T_1/x_1, \ldots, T_n/x_n]$ for $\mathit{Subst}(T, [T_1/x_1, \ldots, T_n/x_n])$. We write Var_s and Term_s for the sets of variables and terms of sort s, $\{x \in \mathit{Var}.\ \mathit{sort}(x) = s\}$ and $\{T \in \mathit{Term}.\ \mathit{sort}(T) = s\}$, respectively. The next lemma lists some basic properties of terms belonging to a term syntax:

Lemma 2.2. *The following hold for all $x, y, z \in \mathit{Var}$ and $T, T' \in \mathit{Term}$:*
(1) If $x \notin \mathit{FV}(T)$, then $T[T'/x] = T$;[1]
(2) $y[T/x] = T$ if $y = x$ and $y[T/x] = y$ otherwise;
(3) If $x \in \mathit{FV}(T)$, then $\mathit{FV}(T[T'/x]) = \mathit{FV}(T) \setminus \{x\} \cup \mathit{FV}(T')$;
(4) If $y \notin \mathit{FV}(T)$, then $T[y/x][z/y] = T[z/x]$;
(5) If $y \notin \mathit{FV}(T)$, then $T[y/x][x/y] = T$.

Definition 2.3. *A* term-generic language *is a tuple $(S, \mathit{Var}, \mathit{sort}, \mathit{Term}, \mathit{FV}, \mathit{Subst}, \Pi)$, where $(\mathit{Term}, \mathit{sort}, \mathit{FV}, \mathit{Subst})$ is a term syntax over (S, Var) and $\Pi = (\Pi_w)_{w \in S^*}$ is an S^*-ranked set of* relation symbols*, with each Π_w being at most countable.*

Therefore, a term-generic language is similar to a standard first-order language, except that the ranked set of operation symbols is replaced by the more abstract notion of term syntax.

2.2. Models

TGL models are structures that interpret terms relative to variable valuations, like Henkin models, and interpret relation symbols standardly. The term interpretation is required to behave well w.r.t. variables and substitution.

Definition 2.4. *A* model *for a term-generic language $(S, \mathit{Var}, \mathit{sort}, \mathit{Term}, \mathit{FV}, \mathit{Subst}, \Pi)$ is a tuple $\mathcal{A} = (A, \mathit{Asort}, (A_T)_{T \in \mathit{Term}}, (A_{(w,\pi)})_{w \in S^*, \pi \in \Pi_w})$, where:*
(a) A is the carrier set *and $\mathit{Asort} : A \to S$ is its sorting function.*

[1] Here and elsewhere: when indicating a substitution of a term for a variable, we implicitly assume that the term's sort and the variable's sort coincide—here, this means $\mathit{sort}(T') = \mathit{sort}(x)$.

(b) For each $T \in Term$, A_T is a mapping $A^{Var,Asort} \to A$, where $A^{Var,Asort}$ is the set of sort-preserving valuations from Var to A, $\{\rho : Var \to A.\ \forall x \in Var.\ Asort(\rho(x)) = sort(x)\}$, such that the following hold for all $x \in Var$, $T \in Term$, $\rho, \rho' \in A^{Var,Asort}$, and $\theta \in Term^{Var,sort}$:
— *(b.i)* $Asort(A_T(\rho)) = sort(T)$;
— *(b.ii)* $A_x(\rho) = \rho(x)$;
— *(b.iii)* $A_{Subst(T,\theta)}(\rho) = A_T(A_\theta(\rho))$, *where* $A_\theta : A^{Var,Asort} \to A^{Var,Asort}$ *is defined by* $A_\theta(\rho)(y) = A_{\theta(y)}(\rho)$.
(c) For each $w = s_1 \dots s_n$ and $\pi \in \Pi_w$, $A_{(w,\pi)} \subseteq \prod_{i \in \overline{1,n}} \{a \in A.\ Asort(a) = s_i\}$.

Thus, unlike for classic FOL models where the interpretation of terms is *built* from interpretations of operation symbols, for TGL models the interpretation of terms is *assumed*, in the style of Henkin models [29].

The model axioms (b.i), (b.ii) and (b.iii) are analogous to the substitution axioms (0), (1) and (4) in Definition 2.1. We can also prove a fact about models analogous to the free-variable axiom (3):[2]

Lemma 2.5. *If $\rho, \rho' \in A^{Var,Asort}$ and $\rho\restriction_{FV(T)} = \rho'\restriction_{FV(T)}$, then $A_T(\rho) = A_T(\rho')$.*

Proof. Thanks to Definition 2.1.(0), we can enumerate each Var_s. We do this emphasizing the free variables of T, obtaining numbers n_s and families $(x_i^s)_{i \in \overline{1,n_s}}$ and $(u_i^s)_{i \in \mathbb{N}}$ such that the following hold for each sort s:
- $Var_s = \{x_i^s.\ i \in \overline{1,n_s}\} \cup \{u_i^s.\ i \in \mathbb{N}\}$;
- $u_i^s \neq u_j^s$ for all $i \neq j$;
- $\{x_i^s.\ i \in \overline{1,n_s}\} \cap \{u_i^s.\ i \in \mathbb{N}\} = \emptyset$;
- $\{y \in FV(T).\ sort(y) = s\} = \{x_i^s.\ i \in \overline{1,n_s}\}$.

For each $s \in S$ and $i \in \mathbb{N}$, we let $v_i^s = u_{2*i}^s$ and $w_i^s = u_{2*i+1}^s$. We define $\Omega, \Omega' : Var \to Var$ by $\Omega(x_i^s) = \Omega'(x_i^s) = x_i^s$, $\Omega(u_i^s) = v_i^s$ and $\Omega'(u_i^s) = w_i^s$, and $\theta, \theta' : Var \to Term$ by $\theta(y) = \Omega(y)$ and $\theta'(y) = \Omega'(y)$. We define $\rho'' : Var \to A$ by $\rho''(x_i^s) = \rho(x_i^s)$, $\rho''(v_i^s) = \rho(u_i^s)$ and $\rho''(w_i^s) = \rho(u_i^s)$. Directly from the definitions and the assumptions, we obtain that these functions are well-sorted and the following hold: (I) $\rho = \rho'' \circ \Omega$ and $\rho' = \rho'' \circ \Omega'$, (II) $\theta\restriction_{FV(T)} = \theta'\restriction_{FV(T)}$. From (II) and Definition 2.1.(3), we obtain: (III) $Subst(T, \theta) = Subst(T, \theta')$. Using Definition 2.4.(b.ii) and (I), we have $A_\theta(\rho'')(y) = A_{\theta(y)}(\rho'') = A_{\Omega(y)}(\rho'') = \rho''(\Omega(y)) = \rho(y)$, obtaining: (IV) $A_\theta(\rho'') = \rho$, and, similarly, $A_{\theta'}(\rho'') = \rho'$.

We now have the following chain of equalities: $A_T(\rho) =$(by (IV))$= A_T(A_\theta(\rho''))$ =(by Definition 2.4.(b.iii)) $= A_{Subst(T,\theta)}(\rho'') =$ (by (III)) $= A_{Subst(T,\theta')}(\rho'') = \dots = A_{T'}(\rho)$. □

In light of the above correspondence between the syntactic and semantic axioms, it should not come as a surprise that the terms can be organized as a model:

Proposition 2.6. *For any relation interpretations $(Term_{(w,\pi)})_{w \in S^*, \pi \in \Pi_w}$ on terms, $(Term, sort, (\rho \mapsto Subst(T, \rho))_{T \in Term}, (Term_{(w,\pi)})_{w \in S^*, \pi \in \Pi_w})$ is a model.*

[2] We are indebted to one of the conference version referees for pointing out that this property needs not be stated as an axiom.

Proof. The conditions of Definition 2.4 follow immediately from the axioms of substitution in Definition 2.1: (b.i) from (0), (b.ii) from (1), and (b.iii) from (4). □

Any model as above will be called a *term model*.

2.3. Formulas and Satisfaction

We fix a term-generic language $(S, Var, sort, Term, FV, Subst, \Pi)$. Formulas, ranged over by φ, ψ, χ, are defined as usual, starting from atomic formulas $\pi(T_1, \ldots, T_n)$ with $\pi \in \Pi_{s_1 \ldots s_n}$ and $sort\,(T_i) = s_i$ and applying the connectives $\wedge, \Rightarrow$ and the quantifier $\forall$. *Formula* denotes the set of formulas.

For each formula φ and model $\mathcal{A}$, the set $A_\varphi \subseteq A^{Var, Asort}$, of valuations that make φ true in $\mathcal{A}$, is defined recursively as follows: if $\pi \in \Pi_w$, $\rho \in A_{\pi(T_1,\ldots,T_n)}$ iff $(A_{T_1}(\rho), \ldots, A_{T_n}(\rho)) \in A_{(w,\pi)}$; $\rho \in A_{\varphi \wedge \psi}$ iff $\rho \in A_\varphi$ and $\rho \in A_\psi$; $\rho \in A_{\varphi \Rightarrow \psi}$ iff $\rho \in A_\varphi$ implies $\rho \in A_\psi$; $\rho \in A_{\forall x.\varphi}$ iff $\rho[x \leftarrow a] \in A_\varphi$ for all $a \in A$ such that $Asort(a) = sort(x)$. If $\rho \in A_\varphi$ we say that A *satisfies* φ *under valuation* ρ and write $\mathcal{A} \models_\rho \varphi$. If $A_\varphi = A^{Var, Asort}$, we say that $\mathcal{A}$ *satisfies* φ and write $\mathcal{A} \models \varphi$. For a set of formulas E, $\mathcal{A} \models E$ means $\mathcal{A} \models \varphi$ for all $\varphi \in \Gamma$.

For formulas, the notions of free variables $FV(\varphi)$, α-equivalence $\varphi \equiv_\alpha \psi$, and (capture-free) substitution $Subst\,(\varphi, \theta)$ are the natural ones, defined similarly to FOL, but on top of our generic terms rather than FOL terms (see Appendix A). For substitution in formulas we adopt notational conventions similar to those for substitution in terms, e.g., $\varphi[T/x]$, assuming $sort\,(T) = sort\,(x)$. A variable x is said to be *fresh* for an item K (where K may be a term, a formula, a set, list or tuple of such items, etc.) if it does not belong to the set of free variables of K. An item K whose set of free variables is finite is said to have *finite support*. A *sentence* is a formula with no free variables. A *theory* is a set of sentences.

Thus, TGL is a logic generic only w.r.t. terms—formulas are "concrete" first-order formulas built over generic terms, with a concrete notion of α-equivalence standardly defined using the $\forall$ binding. The next proposition shows that our axiomatization of generic terms and models is sufficient for deriving TGL analogues of well-known properties of FOL formulas and satisfaction.

Lemma 2.7. *The following hold for all models $\mathcal{A}$, variables x, y, z, terms T, formulas $\varphi, \chi, \varphi', \chi'$, maps $\theta, \theta' \in Term^{Var,sort}$, and valuations $\rho, \rho' \in A^{Var,Asort}$:*
(1) If $\rho{\restriction_{FV(\varphi)}} = \rho'{\restriction_{FV(\varphi)}}$, then $\rho \in A_\varphi$ iff $\rho' \in A_\varphi$;
(2) $\rho \in A_{Subst\,(\varphi,\,\theta)}$ iff $A_\theta(\rho) \in A_\varphi$;
(3) If $\varphi \equiv_\alpha \psi$, then $A_\varphi = A_\psi$;
(4) If $\varphi \equiv_\alpha \psi$, then $FV(\varphi) = FV(\psi)$;
(5) $\equiv_\alpha$ is an equivalence;
(6) $\varphi \equiv_\alpha Subst\,(\varphi, 1_{Var})$;
(7) If $y \notin FV(\varphi)$, then $\varphi[y/x][z/y] \equiv_\alpha \varphi[z/x]$;
(8) If $x \notin FV(\varphi)$, then $\varphi[T/x] \equiv_\alpha \varphi$;
(9) If $\varphi \equiv_\alpha \psi$, then $Subst\,(\varphi, \theta) \equiv_\alpha Subst\,(\psi, \theta)$;
(10) If $\theta{\restriction_{FV(\varphi)}} = \theta'{\restriction_{FV(\varphi)}}$, then $Subst\,(\varphi, \theta) \equiv_\alpha Subst\,(\varphi, \theta')$;
(11) $Subst\,(\varphi, \theta\,; \theta') \equiv_\alpha Subst\,(Subst\,(\varphi, \theta), \theta')$;

(12) If $\varphi \equiv_\alpha \varphi'$ *and* $\psi \equiv_\alpha \psi'$*, then* $\varphi \wedge \psi \equiv_\alpha \varphi' \wedge \psi'$*,* $\varphi \Rightarrow \psi \equiv_\alpha \varphi' \Rightarrow \psi'$*,* $\forall x.\varphi \equiv_\alpha \forall x.\varphi'$*.*

Thus, $\equiv_\alpha$ is an equivalence, preserves satisfaction and the free variables, and is compatible with substitution and the logical connectives (points (5), (3), (4), (9), (12) above). The mappings FV, *Subst*, $A_$ and the logical connectives are therefore well-defined on equivalence classes. Hereafter we identify formulas modulo α-equivalence.

2.4. TGL with equality

A *term-generic language with equality* is a term-generic language with an emphasized binary relation symbol "=", interpreted in all models as the identity relation. Thus, we have atomic formulas $T_1 = T_2$ with $sort\,(T_1) = sort\,(T_2)$, and the following model interpretation: $\rho \in A_{T_1=T_2}$ iff $A_{T_1}(\rho) = A_{T_2}(\rho)$. All the other concepts remain the same.

3. Gentzen System and Completeness

The axiomatic properties of the generic notions of free variable and substitution in TGL provide enough infrastructure to obtain generic versions of classic FOL results. We are interested in a completeness theorem here (but other model-theoretic results could also be generalized from FOL). We will use a generalization of the cut-free system G3c in [49, page 77] (the same as system G from [16, page 187]).

We fix a term-generic language $(S, Var, sort, Term, FV, Subst, \Pi)$. A *sequent* is a pair written $\Gamma \triangleright \Delta$, with the *antecedent* Γ and the *succedent* Δ (at most) countable finite-support sets of formulas. The sequent $\Gamma \triangleright \Delta$ is called *tautological*, written $\models \Gamma \triangleright \Delta$, if $\bigcap_{\varphi\in\Gamma} A_\varphi \subseteq \bigcup_{\psi\in\Delta} A_\psi$ for all models $\mathcal{A}$; it is called *E-tautological* (where E is a set of sentences), written $E \models \Gamma \triangleright \Delta$, if $\mathcal{A} \models E$ implies $\bigcap_{\varphi\in\Gamma} A_\varphi \subseteq \bigcup_{\psi\in\Delta} A_\psi$ for all models $\mathcal{A}$. If $\Gamma = \emptyset$, we write $E \models \Delta$ instead of $E \models \Gamma \triangleright \Delta$.

We will write Γ, Γ' for $\Gamma \cup \Gamma'$ and Γ, φ for $\Gamma \cup \{\varphi\}$. We consider the Gentzen system $\mathcal{G}$ given by the rule schemas in Figure 1, aimed at deducing tautological sequents in TGL. We write $\vdash_{\mathcal{G}} \Gamma \triangleright \Delta$ to mean that $\Gamma \triangleright \Delta$ is deducible in $\mathcal{G}$. If E is a set of sentences, we write $E \vdash_{\mathcal{G}} \Gamma \triangleright \Delta$ for $\vdash_{\mathcal{G}} (E, \Gamma) \triangleright \Delta$ and $E \vdash_{\mathcal{G}} \Delta$ for $E \vdash_{\mathcal{G}} \emptyset \triangleright \Delta$.

Theorem 3.1. *$\mathcal{G}$ is sound and complete for TGL.*

Proof. Soundness: We need to check that each rule is sound. We only consider the quantifier rules, since the soundness of the others is immediate. Let $\mathcal{A}$ be a model. For the soundness of $(\mathsf{Left}\forall)$, it suffices to show that $A_{\forall x.\varphi} \subseteq A_{\varphi[T/x]}$, which is true because of the following: $\rho \in A_{\forall x.\varphi}$ is equivalent to $\rho[x \leftarrow a] \in A_\varphi$ for all $a \in A$, which implies $\rho[x \leftarrow A_T(\rho)] \in A_\varphi$, which in turn is equivalent, by Lemma 2.7.(2) and Definition 2.4.(b.ii), to $\rho \in A_{\varphi[T/x]}$.

$$\frac{\Gamma \cap \Delta \neq \emptyset}{\Gamma \triangleright \Delta}\ (\mathsf{Ax})$$

$$\frac{\Gamma \triangleright \Delta, \varphi \quad \Gamma, \psi \triangleright \Delta}{\Gamma, \varphi \Rightarrow \psi \triangleright \Delta}\ (\mathsf{Left}\Rightarrow) \qquad \frac{\Gamma, \varphi \triangleright \Delta, \psi}{\Gamma \triangleright \Delta, \varphi \Rightarrow \psi}\ (\mathsf{Right}\Rightarrow)$$

$$\frac{\Gamma, \varphi, \psi \triangleright \Delta}{\Gamma, \varphi \wedge \psi \triangleright \Delta}\ (\mathsf{Left}\wedge) \qquad \frac{\Gamma \triangleright \Delta, \varphi \quad \Gamma \triangleright \Delta, \psi}{\Gamma \triangleright \Delta, \varphi \wedge \psi}\ (\mathsf{Right}\wedge)$$

$$\frac{\Gamma, \forall x.\varphi, \varphi[T/x] \triangleright \Delta}{\Gamma, \forall x.\varphi \triangleright \Delta}\ (\mathsf{Left}\forall) \qquad \frac{\Gamma \triangleright \Delta, \varphi[y/x]}{\Gamma \triangleright \Delta, \forall x.\varphi}\ (\mathsf{Right}\forall)\ [y \text{ fresh}]$$

Figure 1: The Gentzen System $\mathcal{G}$

For the soundness of (Right$\forall$), assume $\bigcap_{\chi\in\Gamma} A_\chi \subseteq (\bigcup_{\psi\in\Delta} A_\psi) \cup A_{\varphi[y/x]}$, where y is not free in $\Gamma, \Delta, \forall x.\varphi$. We need to show $\bigcap_{\chi\in\Gamma} A_\chi \subseteq (\bigcup_{\psi\in\Delta} A_\psi) \cup A_{\forall x.\varphi}$. For this, let $\rho \in \bigcap_{\chi\in\Gamma} A_\chi$ such that $\rho \notin \bigcup_{\psi\in\Delta} A_\psi$. We show that $\rho \in A_{\forall x.\varphi}$, i.e., that $\rho[x \leftarrow a] \in A_\varphi$ for all $a \in A$. Let $a \in A$. Because $\rho[y \leftarrow a]\restriction_{FV(\chi)} = \rho\restriction_{FV(\chi)}$ for each $\chi \in \Gamma \cup \Delta$ (since $y \notin FV(\chi)$), by Lemma 2.7.(1), we have $\rho[y \leftarrow a] \in \bigcap_{\chi\in\Gamma} A_\chi$ and $\rho[y \leftarrow a] \notin \bigcup_{\psi\in\Delta} A_\psi$. Thus $\rho[y \leftarrow a] \in A_{\varphi[y/x]}$, hence, by Lemma 2.7.(2), $\rho[y \leftarrow a][x \leftarrow A_{\rho[y\leftarrow a]}(y)] \in A_\varphi$, hence, by Definition 2.4.(b.ii), $\rho[y \leftarrow a][x \leftarrow a] \in A_\varphi$. If $y = x$, we obtain $\rho[x \leftarrow a] \in A_\varphi$, as desired. If $y \neq x$, then $\rho[y \leftarrow a][x \leftarrow a] = \rho[x \leftarrow a][y \leftarrow a]$, hence $\rho[x \leftarrow a][y \leftarrow a] \in A_\varphi$; and since $\rho[x \leftarrow a][y \leftarrow a]\restriction_{FV(\varphi)} = \rho[x \leftarrow a]\restriction_{FV(\varphi)}$, we obtain $\rho[x \leftarrow a] \in A_\varphi$, as desired.

Completeness: The proof mainly follows a classical line (see [16, page 214]). We give the proof in a fair amount of detail, indicating the places where the term-generic axioms are being used.

A sequent $\Gamma \triangleright \Delta$ is called *solved*, if $\Gamma \cap \Delta \neq \emptyset$ (so that (Ax) proves it), and *hopeless*, if it is not solved and no rule can be applied backwards to it. Note that, in a hopeless sequent, both Γ and Δ consist of atomic formulas.

Assume that $\Gamma \triangleright \Delta$ is a tautological sequent. As in [16], we construct backwards from $\Gamma \triangleright \Delta$ a possibly infinite partial proof tree P, roughly by continuously expanding its goals that are neither solved nor hopeless with a fair application of the rules. Special care is required for (Left$\forall$), since when this rule's turn comes according to the considered fair scheduler,[3] a counter n associated to the corresponding formula in Γ needs to be increased, and the rule needs to be applied for each of the first n terms. Moreover, dovetailing needs to be applied to the elements of Γ and Δ, provided these sets are infinite: one fixes an order on the set, and first considers the first element, then the first two elements, etc.

[3]It is essential for the proper behavior of this fair scheduler that the rule (Right$\forall$) be continuously (backwards) enabled for $\forall$-formulas in succedents—this is where our assumption that the succedents Δ have finite support is required; the antecedents Γ also need to have finite support, since they feed the succedents by backwards applications of (Left$\Rightarrow$).

The details are provided in [16]—what is important to notice is that the details of this construction are all *independent of the concrete syntax of terms*.

If P is finite and all its leaves are solved, then we have a (total) proof tree, i.e., a proof of $\Gamma \triangleright \Delta$, as desired. If P is finite and contains a hopeless leaf, say, falsifiable by some valuation ρ in a model $\mathcal{A}$, then, since the rules in $\mathcal{G}$ hold in an "iff" form, it follows that $\Gamma \triangleright \Delta$ is itself falsifiable by ρ in $\mathcal{A}$.

It remains to consider the case of P being infinite. We prove that $\Gamma \triangleright \Delta$ is falsifiable. We choose an infinite path in P, $(\Gamma_i \triangleright \Delta_i)_{i \in I\!N}$; then $\Gamma_0 = \Gamma$ and $\Delta_0 = \Delta$. Let $H_{left} = \bigcup_{i \in I\!N} \Gamma_i$ and $H_{right} = \bigcup_{i \in I\!N} \Delta_i$. By the fair construction of P, we obtain the following:
- $H_{left} \cap H_{right} \cap \mathit{Atomic\ formulas} = \emptyset$;
- $\varphi \wedge \psi \in H_{left}$ implies $\varphi \in H_{left}$ and $\psi \in H_{left}$;
- $\varphi \wedge \psi \in H_{right}$ implies $\varphi \in H_{right}$ or $\psi \in H_{right}$;
- $\varphi \Rightarrow \psi \in H_{left}$ implies $\varphi \in H_{right}$ or $\psi \in H_{left}$;
- $\varphi \Rightarrow \psi \in H_{right}$ implies $\varphi \in H_{left}$ and $\psi \in H_{right}$;
- $\forall x.\varphi \in H_{left}$ implies for all $T \in \mathit{Term}$, $\varphi[T/x] \in H_{left}$;
- $\forall x.\varphi \in H_{right}$ implies there exists y with $\varphi[y/x] \in H_{right}$.

In order to falsify $\Gamma \triangleright \Delta$, it suffices to falsify $H_{left} \triangleright H_{right}$. We take $\mathcal{A}$ to be a Herbrand-like term model (as in Proposition 2.6), defining the relation interpretations by $(T_1, \ldots, T_n) \in A_{w,\pi}$ iff $\pi(T_1, \ldots, T_n) \in H_{left}$.

We show that $H_{left} \triangleright H_{right}$ is falsified by the valuation ι, the inclusion of variables into terms. We prove by induction on the depth of φ the following statement: $[\varphi \in H_{left}$ implies $\iota \in A_\varphi]$ and $[\varphi \in H_{right}$ implies $\iota \notin A_\varphi]$.

If φ has the form $\pi(T_1, \ldots, T_n)$, then by the definition of $\mathcal{A}$ and Definition 2.1.(2), $\iota \in A_{\pi(T_1,\ldots,T_n)}$ iff $\pi(A_{T_1}(\iota), \ldots, A_{T_n}(\iota)) \in H_{left}$ iff $\pi(T_1, \ldots, T_n) \in H_{left}$. In particular, $\pi(T_1, \ldots, T_n) \in H_{left}$ implies $\iota \in A_{\pi(T_1,\ldots,T_n)}$. Moreover, since $\pi(T_1, \ldots, T_n)$ is atomic and $H_{left} \cap H_{right} \cap \mathit{Atomic\ formulas} = \emptyset$, it cannot happen that $\pi(T_1, \ldots, T_n) \in H_{right}$ and $\iota \in A_{\pi(T_1,\ldots,T_n)}$, thus $\pi(T_1, \ldots, T_n) \in H_{right}$ implies $\iota \notin A_{\pi(T_1,\ldots,T_n)}$.

The case of the logical connectives $\wedge$ and $\Rightarrow$ is straightforward.

Assume now φ has the form $\forall x.\psi$. From $\forall x.\psi \in H_{left}$ we infer that $\psi[T/x] \in H_{left}$ for each term T (i.e., for each element T of A), and furthermore, by the induction hypotheses, that $\iota \in A_{\psi[T/x]}$, i.e., $\iota \in A_{\psi[T/x]}$, i.e., by Lemma 2.7.(2), $\iota[x \leftarrow A_\iota(T)] \in A_\psi$, i.e., by Definition 2.4.(b.(ii)), $\iota[x \leftarrow T] \in A_\varphi$, for each $T \in A$; hence $\iota \in A_{\forall x.\psi}$. That $\forall x.\psi \in H_{right}$ implies $A \not\models_\iota \forall x.\psi$ can be proved similarly, again by Lemma 2.7.(2) and Definition 2.4.(b.(ii)). □

An interesting question is whether the syntax axioms (in Definition 2.1) and the model axioms (in Definition 2.4) are necessary for the development of a substantial FOL-like theory of TGL, including the above soundness and completeness theorem. Next we give a partial answer to this question. Note that there is a fine balance between these two types of axioms. As a first approximation, we have the following monotonicity criterion: the fewer syntax axioms and/or more model axioms, the more likely that soundness holds; the more syntax axioms and/or fewer model axioms, the more likely that completeness holds. (Soundness constrains the models, completeness constrains the syntax.)

Thus, the above completeness proof would not be possible without the term models offered by Proposition 2.6, whose proof requires that the model axioms be matched by corresponding syntax axioms. Also, soundness of the $\forall$-rules requires the model axioms (b.ii) and (b.iii), with (Right$\forall$) also requiring Lemma 2.5—(b.iii) and Lemma 2.5 are used indirectly, via their corresponding properties for formulas, Lemma 2.7.(2,1).

But there are subtle exceptions to this criterion. Completeness also requires the syntactic axiom (2) to properly interpret atomic formulas in the term model. Moreover, the ability to always pick a fresh y enabling the rule (Right$\forall$), crucial for completeness, is ensured by the syntactic axiom (0). Finally, Lemma 2.5, used for soundness, requires the syntax axiom (3) concerning free variables—of course, the property of Lemma 2.5 could instead be postulated as an axiom, and then Proposition 2.6, hence the completeness proof, would require the axiom (3). Therefore, not only that the monotonicity criterion is not entirely accurate, but also the equilibrium between what is required for soundness and what for completeness is affected by the choice of the axioms.

Outside the soundness-completeness dichotomy, the development presented in Appendix B shows that the syntactic axioms are all invoked in proving common-sense properties of formulas, such as satisfaction being invariant under renaming of bound variables and substitution on formulas behaving well.

Therefore, it seems probable that any substantial development of a FOL-like logic based on substitution and free variables should either assume or be able to prove these fundamental properties. Substitution-compositional recursors for terms with bindings, including those involved in higher-order abstract syntax representations, also appear to require these properties [38].

To prove soundness and completeness for TGL with equality, we enrich $\mathcal{G}$ with the equality rules from Figure 2, obtaining the Gentzen system $\mathcal{G}_{=}$. Since the equality rules can be simulated in the Gentzen $\mathcal{G}$ by adding axioms to the antecedents of the sequents, we obtain:

Theorem 3.2. *$\mathcal{G}^{=}$ is sound and complete for TGL with equality.*

From now on, we write $\vdash_{TGL} \Gamma \triangleright \Delta$ for deduction in TGL, the context always clarifying whether we mean $\mathcal{G}$ or $\mathcal{G}_{=}$. For E set of sentences, we write $E \vdash_{TGL} \Gamma \triangleright \Delta$ for $\vdash_{TGL} (E, \Gamma) \triangleright \Delta$ and $E \vdash_{TGL} \Delta$ for $E \vdash_{TGL} \emptyset \triangleright \Delta$.

4. Defining Systems in Term-Generic Logic

We start with a brief general discussion concerning the typical pattern of rules in our specified calculi and how we go about representing them in TGL (Subsection 4.1). Then we consider some concrete TGL specifications, where the reader should easily be able to track the described pattern: System F (Subsection 4.2), the π-calculus (Subsection 4.3) and the equational versions of untyped λ-calculus and System F (Subsection 4.4).

$$\frac{\Gamma, T = T \rhd \Delta}{\Gamma \rhd \Delta} \quad \text{(R)}$$

$$\frac{\Gamma \rhd \Delta, T_1 = T_2 \qquad \Gamma, T_2 = T_1 \rhd \Delta}{\Gamma \rhd \Delta} \quad \text{(S)}$$

$$\frac{\Gamma \rhd \Delta, T_1 = T_2 \qquad \Gamma \rhd \Delta, T_2 = T_3 \qquad \Gamma, T_1 = T_3 \rhd \Delta}{\Gamma \rhd \Delta} \quad \text{(T)}$$

$$\frac{\begin{array}{c}\Gamma \rhd \Delta, T_1 = T'_1 \quad \ldots \quad \Gamma \rhd \Delta, T_n = T'_n \\ \Gamma \rhd \Delta, \pi(T_1, \ldots, T_n) \qquad \Gamma, \pi(T'_1, \ldots, T'_n) \rhd \Delta\end{array}}{\Gamma \rhd \Delta} \quad (\text{Cmp}_\pi)$$

$$\frac{\Gamma \rhd \Delta, T_1 = T_2 \qquad \Gamma, T[T_1/x] = T[T_2/x] \rhd \Delta}{\Gamma \rhd \Delta} \quad \text{(Sbs)}$$

Figure 2: The equality rules (and $\mathcal{G}_= = \mathcal{G}$ + the equality rules)

4.1. General Remarks

The calculi we discuss have typical deduction rules of the form

$$\frac{\Gamma, h(\overline{x}, \overline{T}) \rhd c(\overline{T}, \overline{T'})}{\Gamma \rhd d(f(\overline{x}, \overline{T'}), \overline{T})} \quad \begin{array}{l}\textsf{(Orig)} \\ [\overline{x} \text{ fresh for } \Gamma]\end{array}$$

where, in d, $\overline{x}$ are assumed bound in $\overline{T'}$ by some syntactic term operator f.[4]

Such rules are modeled in TGL roughly as follows: (1) the relationship between the assumption and the conclusion of the rule becomes implication; (2) the relationship between the antecedent and the succedent of the sequent also becomes implication; (3) the variables assumed fresh become universally quantified, having as scope the implication from (2); (4) the terms with no variables assumed bound to them become variables; (5) the part of the antecedent common to the hypothesis and the conclusion is removed (since it is handled implicitly by the TGL context). All in all, the rule (Orig) of the original calculus becomes in TGL an axiom-schema parameterized by the terms $\overline{T'}$:

$$\begin{array}{l}(\forall \overline{x}.\ h(\overline{x}, \overline{y}) \Rightarrow c(\overline{y}, \overline{T'})) \\ \Rightarrow\ d(f(\overline{x}, \overline{T'}), \overline{y})\end{array} \quad \textsf{(Rep)}$$

We have developed a systematic syntactic technique to establish adequacy of TGL representations (not included in this paper, but presented in detail in the technical report [40]). In a nutshell, the technique "reverse-engineers" the TGL schemas (Rep) into a format, say (Rep'), that resembles the "original" rule (Orig) and hence essentially reduces adequacy to noticing isomorphism between two deduction trees: one in the original calculus, and one in a customized

[4]This is a typical pattern, not an exhaustive one; rules may have multiple assumptions, the additional hypothesis $h(\overline{x}, \overline{T})$ may lack from the lefthand side of assumptions, etc.

Gentzen system for TGL. (Rep') is obtained from (Rep) by applying the TGL left rules for implication and universal quantification and has the same shape as (Orig), but with Γ consisting of TGL atomic assumptions rather than assumptions/hypotheses in the original calculus. In order for the aforementioned reduction to be soundly and completely applicable, a proof-theoretic condition (often reducible to a mere syntactic check) needs to be verified for the TGL specification.

Conventions. The following convention regarding meta-variables for variables and for terms in formula schemas, although standard in FOL, needs to be recalled here to avoid the misreading of the forthcoming TGL specifications. Let us assume that the set of variables *Var* is $v_0, v_1, v_2, \ldots$, and x, y, z, z' have been set as symbols (i.e., meta-variables) ranging over variables. Let R be a binary relation symbol. When we write, for instance, $\forall x, y.\ R(x, y) \Rightarrow R(y, x)$, we mean a single sentence obtained by a *choice* of the *distinct* variables x and y, e.g., $x = v_0$ and $y = v_1$. This choice is, of course, semantically immaterial. (In fact, since we identify formulas modulo α-equivalence, it is even syntactically immaterial.)

Now, assume we work with a TGL language having as term syntax that of the untyped λ-calculus (with the terms taken to be α-classes), and a binary relation symbol R. A schema written as

$$R(x, y) \wedge (\forall z.\ R(X, Y)) \Rightarrow R((\lambda z.\, X)\, x, (\lambda z.\, Y)\, y)$$

is interpreted as follows: we choose (and fix) three *distinct* variables x, y, z; then the schema is read as a set of formulas as above, one formula for each pair of terms (X, Y). X and Y can very well make free use of the variables x, y, and z—this is usually crucial for the intended meaning of the schema. Note that it makes sense to speak of potential (free) occurrences of z in X, since this notion is well-defined for α-classes. Likewise, it makes sense to say that the operator λ binds z in X, since the operator λz is well-defined on α-classes.

4.2. System F

System F is an impredicative polymorphic typed λ-calculus [17, 42]. Its syntax modulo α-equivalence forms a two-sorted TGL term syntax. The sorts are *type* and *data*, and we let t range over Var_{type}, x, y over Var_{data}, T over $Term_{type}$, and X, Y over $Term_{data}$. Here is the grammar for terms, or, more precisely, for the raw terms out of which, by quotienting to the standard notion of α-equivalence, one obtains the terms:

$$\begin{array}{lcl} T & ::= & t \mid T \to T' \mid \Pi\, t.\, T \\ X & ::= & x \mid \lambda\, x{:}T.\, X \mid X\, Y \mid \lambda\, t.\, X \mid X\, T \end{array}$$

(Sometimes, one writes $\forall\, t.\, T$ instead of $\Pi\, t.\, T$ and $X[T]$ instead of $X\, T$.) The bindings are as expected: in $\lambda\, x{:}T.\, X$, x is bound in X; in $\lambda\, t.\, X$, t is bound in X; in $\Pi\, t.\, T$, t is bound in T. A *typing context* Γ is a finite set $\{x_1{:}T_1, \ldots, x_n{:}T_n\}$, where the x_i's are data variables, the T_i's are type terms, and $x_i \neq x_j$ for $i \neq j$.

$\Gamma \triangleright x{:}T$ (SF-InVar) $[(x{:}T) \in \Gamma]$	
$\dfrac{\Gamma, x{:}T \triangleright X : T'}{\Gamma \triangleright (\lambda x{:}T.X) : T \to T'}$ (SF-Abs) $[x$ fresh for $\Gamma]$	$(\forall x.\, tpOf(x,t) \Rightarrow tpOf(X,t')) \Rightarrow tpOf(\lambda x{:}t.X,\, t \to t')$ (Abs)
$\dfrac{\Gamma \triangleright X : T \to T' \quad \Gamma \triangleright Y : T}{\Gamma \triangleright X\,Y : T'}$ (SF-App)	$tpOf(x, t \to t') \wedge tpOf(y,t) \Rightarrow tpOf(x\,y, t')$ (App)
$\dfrac{\Gamma \triangleright X : T}{\Gamma \triangleright (\lambda t.X) : \Pi t.T}$ (SF-T-Abs) $[t$ fresh for $\Gamma]$	$(\forall t.\, tpOf(X,T)) \Rightarrow tpOf(\lambda t.X,\, \Pi t.\, T)$ (T-Abs)
$\dfrac{\Gamma \triangleright X : \Pi t.T}{\Gamma \triangleright X\,T' : T[T'/t]}$ (SF-T-App)	$tpOf(x,\, \Pi t.\, T) \Rightarrow tpOf(x\,t,\, T)$ (T-App)

Figure 3: System F typing: Original system TSF (left) and TGL specification $\mathcal{TSF}$ (right)

The conventional typing system for System F [17, 42], which we call TSF, is shown on the left of Figure 3. It derives sequents of the form $\Gamma \triangleright X : T$.

We specify TSF as a TGL theory over a language having the aforementioned term syntax and one relation symbol, *tpOf*, read "type of", of arity $data \times type$. *tpOf* captures both the assumptions $x : T$ from the typing contexts and the actual typing relationships $X : T$. The TGL theory is shown in Figure 3, on the right. Each typing rule in TSF, except for (SF-InVar), yields an axiom or axiom-schema in $\mathcal{TSF}$ obtained as explained at the beginning of this section, by identifying the implicit universal quantification and implication involved in the intuitive reading of the rule. For example, we read (SF-Abs) as follows: If one can type X to T' *uniformly on* x assuming x has type T, i.e., *for all* x of type T, then $\lambda x : T.\, X$ receives the type $T \to T'$; this is naturally expressed as the TGL sentence (Abs). T and T' from (SF-Abs) are not involved in any bindings relevant here, hence they become the variables t and t' in (Abs).

(Abs), (T-Abs) and (T-App) are axiom-schemas, parameterized by arbitrary terms X, T. Thus, the upper case meta-variables X, T denote parameters (subject to capturing instantiation), unlike the lower case meta-variables x, t, t', which denote fixed variables subject to binding and (capture-free) substitution.

In (Abs), a presumptive occurrence of x in the left X is in the scope of the universal quantifier, and one in the right X is in the scope of the λ-abstraction; similarly for t versus X and t versus T in (T-Abs). This migration of the variables x and t between scopes may look surprising at first. Note however that the same situation appears in the corresponding rules ((SF-Abs) and (SF-T-Abs)) from the original system TSF. Thus, in (SF-Abs), any occurrence of x in X from the succedent of the conclusion $\Gamma \triangleright (\lambda x : T.\, X) : T \to T'$ is in the scope of the λ-abstraction, while the same occurrence of x in X when part of the antecedent of the hypothesis $\Gamma, x : T \triangleright X : T'$ is not in any scope (or can be

$(\lambda x{:}T.\,Y)X \rightsquigarrow Y[X/x]$ (SF-β)	$(\lambda x : t.\,Y)\,x \rightsquigarrow Y$ (β)
$(\lambda t.\,Y)T \rightsquigarrow Y[T/t]$ (SF-T-β)	$(\lambda t.\,Y)t \rightsquigarrow Y$ (T-β)
$\dfrac{X \rightsquigarrow X'}{\lambda x{:}T.\,X \rightsquigarrow \lambda x{:}T.\,X'}$ (SF-ξ)	$(\forall x.\,X \rightsquigarrow X')$ $\Rightarrow \lambda x{:}t.\,X \rightsquigarrow \lambda x{:}t.\,X'$ (ξ)
$\dfrac{X \rightsquigarrow X'}{\lambda t.\,X \rightsquigarrow \lambda t.\,X'}$ (SF-T-ξ)	$(\forall t.\,X \rightsquigarrow X')$ $\Rightarrow \lambda t.\,X \rightsquigarrow \lambda t.\,X'$ (T-ξ)
$\dfrac{X \rightsquigarrow X'}{X\,Y \rightsquigarrow X'\,Y}$ (SF-AppL)	$x \rightsquigarrow x'$ $\Rightarrow x\,y \rightsquigarrow x'\,y$ (AppL)
$\dfrac{Y \rightsquigarrow Y'}{X\,Y \rightsquigarrow X\,Y'}$ (SF-AppR)	$y \rightsquigarrow y'$ $\Rightarrow x\,y \rightsquigarrow x\,y'$ (AppR)
$\dfrac{X \rightsquigarrow X'}{X\,T \rightsquigarrow X'\,T}$ (SF-T-AppC)	$x \rightsquigarrow x'$ $\Rightarrow x\,t \rightsquigarrow x'\,t$ (T-AppC)

Figure 4: System F reduction: Original system RSF (left) and TGL specification $\mathcal{RSF}$ (right)

considered in the scope of the implicit outer binder of the sequent).

It is instructive to analyze what would happen if we neglected the proper scoping of x in the assumption of (Abs), replacing (Abs) by the following:

$$\begin{array}{c}(\mathit{tpOf}(x,t) \Rightarrow \mathit{tpOf}(X,t')) \\ \Rightarrow \mathit{tpOf}(\lambda x{:}t.X,\ t \to t')\end{array} \quad \text{(Abs')}$$

In (Abs'), the variable x from $\mathit{tpOf}(x,t)$ is now, just like the variables t and t' from $\mathit{tpOf}(x,t)$ and $\mathit{tpOf}(X,t')$ respectively, in the scope of some (implicit) outermost universal quantifiers; consequently, x, t and t' can be instantiated with anything, e.g., any data term Y and type terms T and T' such that $\mathit{tpOf}(X,T)$ does not hold—this would render the upper implication trivially true, and hence would allow us to conclude $\mathit{tpOf}(\lambda x{:}t.X,\ T \to T')$. In summary, (Abs') would allow us to type $\lambda x{:}t.X$ to $T \to T'$ provided we can display *any* item Y that does not have type T, which is clearly not what we want. By contrast, (Abs) conditions $\mathit{tpOf}(\lambda x : t.X,\ t \to t')$ by $\forall x.\,\mathit{tpOf}(x,t) \Rightarrow \mathit{tpOf}(X,t')$, i.e., by the possibility to prove $\mathit{tpOf}(x,t) \Rightarrow \mathit{tpOf}(X,t')$ *generically* in x—this prevents the above undesired behavior.

Next, we consider System F's conventional reduction system [17, 42], which we call RSF, and its TGL representation, $\mathcal{RSF}$. They are indicated in Figure 4. The TGL theory is in a language having one binary relation symbol, $\rightsquigarrow$, of arity $\mathit{data} \times \mathit{data}$.

In (β), we employ the same variable x to indicate both the *formal parameter* of the functional expression $\lambda x : t.\,Y$ and its *actual parameter* (the occurrence

of x on the right of the application from the left of $\rightsquigarrow$). Indeed, in the latter case, as well as in any presumptive occurrences in the right Y, x is exposed to an (implicit) outer universal quantifier of the TGL sentence, hence denotes an (arbitrary) actual value in a model.

Next we state adequacy of our representation without proofs, but refer to the technical report [40] for full proofs in the context of the aforementioned systematic adequacy technique.[5] For each typing context $\Gamma = \{x_1{:}T_1, \ldots, x_n{:}T_n\}$, we let $\Gamma^{\#}$ be the set of TGL atomic formulas $\{\mathit{tpOf}(x_1, T_1), \ldots, \mathit{tpOf}(x_n, T_n)\}$.

Proposition 4.1. *The following are equivalent:*
(a) $\vdash_{TSF} \Gamma \triangleright X : T$.
(b) $\mathcal{TSF} \vdash_{TGL} \Gamma^{\#} \triangleright \mathit{tpOf}(X, T)$.

Proposition 4.2. *The following are equivalent:*
(a) $\vdash_{RSF} X \rightsquigarrow Y$.
(b) $\mathcal{RSF} \vdash_{TGL} X \rightsquigarrow Y$.

Thus, adequacy ensures two crucial properties:
- Soundness: Everything deducible in the system (TSF or RSF) is also deducible in TGL from the associated theory ($\mathcal{TSF}$ or $\mathcal{RSF}$)—(a) implies (b);
- Completeness: Everything deducible in TGL from the associated theory *and expressible in the system* is also deducible in the system—(b) implies (a).

Note that our adequacy statements refer to *what* is being deduced, and not to *how* statements are being deduced—a stronger form of adequacy would not only relate the deduced facts in the two systems (here, the original calculus and TGL), but would also establish an isomorphism between their deduction trees. This stronger form does not hold here, at least not w.r.t. the presented Gentzen system $\mathcal{G}$ for TGL, since multiple applications of the primitive rules of $\mathcal{G}$ would be required to handle one rule of the original calculus. In [40], we present a customized system for TGL that ensures strong adequacy.

4.3. The π-Calculus

The π-calculus [28] is often considered non-standard w.r.t. transition-time binding mechanisms due to its scope extrusion feature (via sending bound names). Interestingly, this non-standard aspect vanishes as soon as one regards π-calculus transitions as terms with bindings themselves. Then transitions, together with channel names and processes, form a TGL term syntax. We omit the original presentation of the calculus, but indicate it directly as a TGL theory. We also omit the choice (+) and the replication (!) operators—these operators (included in the extended technical report [40]) do not raise any interesting binding issues.

[5] In Section 5 we obtain semantic proofs of adequacy for the untyped λ-calculus and equational System F as a consequence of our model analysis.

Three sorts: *proc*, *chan*, and *trans*. x, y, z range over Var_{chan} (which will be equal to $Term_{chan}$), X over $Term_{chan}$, p, q, r over Var_{proc}, P, Q, R over $Term_{proc}$, t over Var_{trans}, and T over $Term_{trans}$. Grammar for terms:

$$\begin{array}{lcl} X & ::= & x \\ P & ::= & p \mid 0 \mid P|Q \mid \mathsf{Inp}(x,y,P) \mid \mathsf{Out}(x,y,P) \mid \nu y.\, P \\ T & ::= & t \mid \mathsf{Tau}(P,Q) \mid \mathsf{Inp}(P,x,y,Q) \mid \mathsf{Fout}(P,x,y,Q) \mid \mathsf{Bout}(P,x,y,Q) \end{array}$$

The bindings are as follows: in $\mathsf{Inp}(x,y,P)$ and $\nu y.\, P$, y is bound in P; in $\mathsf{Bout}(P,x,y,Q)$ and $\mathsf{Inp}(P,x,y,Q)$, y is bound in Q. Note that we use Inp to indicate both input prefixing in processes and input-action transitions. Fout and Bout stand for "free output" and "bound output". There are 4 types of possible transitions of a process P, into a process Q: (1) silent transition $\mathsf{Tau}(P,Q)$; (2) input transition $\mathsf{Inp}(P,x,y,Q)$, which receives a generic (bound) input y on channel x; (3) free output transition $\mathsf{Fout}(P,x,y,Q)$, which emits a concrete (free) name y on channel x; (4) bound output transition $\mathsf{Bout}(P,x,y,Q)$, which emits a new (bound) name y on channel x.

We let Op range over $\{\mathsf{Inp}, \mathsf{Fout}, \mathsf{Bout}\}$. The TGL theory is in a language with one relation symbol, *valid*, of arity *trans*. It describes the valid transitions:

$valid(\mathsf{Inp}(\mathsf{Inp}(x,y,P),x,y,P))$	(Inp)
$valid(\mathsf{Fout}(\mathsf{Out}(x,y,P),x,y,P))$	(Out)
$(\forall z.\, valid(Op(P,x,y,P')) \Rightarrow valid(Op(\nu z.\, P,x,y,\nu z.\, P'))$	(Nu)
$(\forall z.\, valid(\mathsf{Tau}(P,P')) \Rightarrow valid(Op(\nu z.\, P,\nu z.\, P'))$	(Nu_τ)
$(\forall y.\, valid(\mathsf{Fout}(P,x,y,P')) \Rightarrow valid(\mathsf{Bout}(\nu y.\, P,x,y,P'))$	(Open)
$valid(Op(p,x,y,P')) \Rightarrow valid(Op(p \mid q,x,y,P' \mid q))$	(Par)
$valid(Op(p,p')) \Rightarrow valid(Op(p \mid q,p' \mid q))$	(Par_τ)
$valid(\mathsf{Fout}(P,x,y,P')) \wedge valid(\mathsf{Inp}(Q,x,y,Q')) \Rightarrow valid(\mathsf{Tau}(P \mid Q,P' \mid Q'))$	(Comm_{free})
$valid(\mathsf{Bout}(P,x,y,P')) \wedge valid(\mathsf{Inp}(Q,x,y,Q')) \Rightarrow valid(\mathsf{Tau}(P \mid Q,\nu y.\, (P' \mid Q')))$	(Comm_{bound})

We omitted the symmetricals of (Par), (Par_τ), (Comm_{free}) and (Comm_{bound}).

Remarks. (1) This theory illustrates the migration of variables not only between the scopes of term bindings and universal quantifiers, but also between the scopes of two term bindings. E.g., in the axiom (Inp), any presumptive occurrence of y in the left P is bound by the *process* binding construct Inp, while the same occurrence of y in the right P is in the (outer) scope of the *transition* binding construct Inp. This models a process having a generic input capability which is consumed in a generic input transition.

(2) In the axiom-schema (Nu), x, y, z are some fixed mutually distinct variables. The fact that z is distinct from x and y is essential for the intended behavior (just as much as it is essential in a FOL axiom such as $\forall x, y, z.\ x = y \wedge y = z \Rightarrow x = z$), ensuring, via the capture-avoidance nature of TGL substitution, that no "concrete instances" of x and y (i.e., channel variables substituting x and y) are allowed to be z, keeping them unaffected by the hider νz, as desired.

(3) A rule such as (Open) traditionally opens the scope of a bound name y in a process P (which becomes P')—in our case, the scope is in effect transferred to the transition binder Bout. Via the axiom (Comm_{bound}), at communication time, this transition binder delivers the scope "safely" back to the hider νy, after another process has joined in. This careful fostering of the scopes (which are never allowed to "escape"), implicit in the underlying logic TGL, saves the trouble of having to state awkward and error-prone side-conditions.

(4) In (Comm_{free}), we have a situation similar to the TGL variant of (β): the instantiation of the input formal parameter with an actual parameter outputted by the communicating process is represented using the same name, y.

Due to our apparently non-standard approach to representing the π-calculus, it is instructive to spell out its deductive closure in TGL, that is, the rules obtained from the TGL axioms by instantiating the top $\forall$-quantified variables by arbitrary terms and writing implications $\varphi \Rightarrow \psi$ in rule form $\frac{\varphi}{\psi}$. In these rules, the reader should recognize the traditional presentation of the calculus.

But first we introduce some notation and terminology. We will use syntactic sugar for atomic TGL formulas corresponding to the 4 kinds of transitions:
$P \xrightarrow{\tau} P'$, for $valid(\mathsf{Tau}(P, P'))$; $P \xrightarrow{x(y)} P'$, for $valid(\mathsf{Inp}(P, x, y, P'))$;
$P \xrightarrow{\overline{x}y} P'$, for $valid(\mathsf{Fout}(P, x, y, P'))$; $P \xrightarrow{\overline{x}(y)} P'$, for $valid(\mathsf{Bout}(P, x, y, P'))$.

A *loud action* is an item having one of the following forms: $x(y), \overline{x}y, \overline{x}(y)$. We let γ range over loud actions. In the actions $\overline{x}(y)$ and $x(y)$, x is considered to appear free and y bound; in the action $\overline{x}y$, both x and y are free. $BV(\gamma)$, $FV(\gamma)$ and $V(\gamma)$ denote the (at most two-element) sets of all the *bound*, *free* and *arbitrary* variables of γ. Note that actions are not identified modulo alpha-equivalence—the notion of a variable being free/bound in an action, although standard, is somewhat misleading, since the scope of the binding is not the action, but rather, as made explicit by our TGL representation, the target process of the transition.

$$\frac{\cdot}{\mathsf{Inp}(x,y,P) \xrightarrow{x(y)} P}\ (\mathsf{Inst\text{-}Inp})$$

$$\frac{\cdot}{\mathsf{Fout}(x,y,P) \xrightarrow{\overline{x}y} P}\ (\mathsf{Inst\text{-}Out})$$

$$\frac{P \xrightarrow{\gamma} P'}{\nu\, z.\, P \xrightarrow{\gamma} \nu\, z.\, P'}\ (\mathsf{Inst\text{-}Nu})\ [z \notin V(\gamma)]$$

$$\frac{P \xrightarrow{\tau} P'}{\nu\, z.\, P \xrightarrow{\tau} \nu\, z.\, P'}\ (\mathsf{Inst\text{-}Nu}_{\tau})$$

$$\frac{P \xrightarrow{\overline{x}y} P'}{\nu\, y.\, P \xrightarrow{\overline{x}(y)} P'}\ (\mathsf{Inst\text{-}Open})\ [x \neq y]$$

$$\frac{P \xrightarrow{\gamma} P'}{P \,|\, Q \xrightarrow{\gamma} P' \,|\, Q}\ (\mathsf{Inst\text{-}Par})\ [BV(\gamma) \cap FV(Q) = \emptyset]$$

$$\frac{P \xrightarrow{\tau} P'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, Q}\ (\mathsf{Inst\text{-}Par}_{\tau})$$

$$\frac{P \xrightarrow{\overline{x}z} P' \qquad Q \xrightarrow{x(y)} Q'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, (Q'[z/y])}\ (\mathsf{Inst\text{-}Comm}_{free})$$

$$\frac{P \xrightarrow{\overline{x}(y)} P' \qquad Q \xrightarrow{x(y)} Q'}{P \,|\, Q \xrightarrow{\tau} \nu y.\, (P' \,|\, Q')}\ (\mathsf{Inst\text{-}Comm}_{bound})$$

The side conditions from the above rules are all explained in terms of TGL substitution/instantiation. For $(\mathsf{Inst\text{-}Nu})$, if x' and y' are the components of the

loud action γ, then they appeared as terms[6] to substitute the variables x and y from the TGL axiom (Nu)—consequently, z has to be fresh for x', y', which means $z \notin V(\gamma)$. The case of the side-condition from (Inst-Open) is similar. As for the side-condition from (Inst-Par), the only nontrivial situation is when γ is $x(y)$ or $\overline{x}(y)$, case in which, in the corresponding TGL axiom (Par), Op is Inp or Bout, meaning that in (Par) the right occurrence of the process variable q is in the scope of the binder Op (which binds y), further meaning that any process term Q substituting q (such as the one from (Inst-Par)) is not allowed to have y as a free variable.

4.4. Equational Calculi

Sometimes a calculus does not come with a reduction relation, but with an equational theory. For example, the original λ-calculus is equational [7]. For these situations, TGL *with equality* is a more appropriate. The TGL-with-equality theories of equational untyped λ-calculus and System F are given below.

Untyped equational λ-Calculus ($U\lambda_=$). An unsorted theory in TGL with equality. x, y, z range over *Var* and X, Y, Z over terms. Grammar for terms: $X ::= x \mid X\,Y \mid \lambda x.\, X$. The bindings are as expected: in $\lambda x.\, X$, x is bound in X. There are no relation symbols (except for equality). Axioms:

$(\forall x.X = Y) \Rightarrow \lambda x.X = \lambda x.Y$	(ξ)
$(\lambda x.X)x = X$	(β)
$x = \lambda y.x\,y$	(η)

We have already discussed the managing of scoping in rules such as (ξ), so the universal quantification over x nested in the assumption should not come as a surprise to the reader. We should only mention that removing this quantification, i.e., replacing (ξ) by $X = Y \Rightarrow \lambda x.X = \lambda x.Y$ would have a "devastating" effect on the specification: taking X to be x and Y to be a variable y different from x, we would obtain $\forall x, y.\, x = y \Rightarrow \lambda x.x = \lambda x.y$; hence (instantiating both x and y to y) $\forall y.\, y = y \Rightarrow \lambda x.x = \lambda x.y$; hence $\forall y.\, \lambda x.x = \lambda x.y$; hence $\forall x, y.\, (\lambda x.x)\, x = (\lambda x.y)\, x$; hence, by (β), $\forall x, y.\, x = y$. (All throughout we implicitly applied TGL's equality, connective and quantifier rules.) In summary, the above flawed version of (ξ), together with (β), collapse all models into the singleton model.

Equational System F ($\mathcal{SF}_=$) —theory over TGL with equality. Two sorts: *type* and *data*. One relation symbol (besides equality), *tpOf*, of arity

[6] Recall that all channel terms are in fact variables.

data × *type*. The terms are the ones of $\mathcal{SF}$. Axioms:

$(\forall x.tpOf(x,t) \Rightarrow tpOf(X,t')) \Rightarrow tpOf(\lambda x\!:\!t.X, t \to t')$	[Abs]
$tpOf(x, t \to t') \wedge tpOf(y,t) \Rightarrow tpOf(x\,y, t')$	[App]
$(\forall t.tpOf(X,T)) \Rightarrow tpOf(\lambda t.X, \Pi t.\,T)$	[T-Abs]
$tpOf(x, \Pi t.\,T) \Rightarrow tpOf(x\,t, T)$	[T-App]
$(\forall x.tpOf(x,t) \Rightarrow X = Y) \Rightarrow \lambda x\!:\!t.X = \lambda x\!:\!t.Y$	(ξ)
$tpOf((\lambda x\!:\!t.X)x, t') \Rightarrow (\lambda x\!:\!t.X)x = X$	(β)
$tpOf(\lambda y\!:\!t.x\,y, t') \Rightarrow x = \lambda y\!:\!t.x\,y$	(η)
$(\forall t.X = Y) \Rightarrow \lambda t.X = \lambda t.Y$	(T-ξ)
$tpOf(\lambda t.X, t') \Rightarrow (\lambda t.X)t = X$	(T-β)
$tpOf(\lambda t.x\,t, t') \Rightarrow x = \lambda t.x\,t$	(T-η)

Thus, the typing axioms are the ones from Subsection 4.2 and the axioms for the equational theory reflect the ones in Subsection 4.2 for reduction, but also ensure that the two terms in the equalities are well-typed—such caution is necessary in any interaction between equational theories and types, as shown, e.g., in [29], where equalities for System F appear in typing contexts and are conditional on the well-typedness of the participants. Axioms of compatibility with application, as well as reflexivity, symmetry and transitivity need not be stated, since they are wired in TGL with equality.

5. Ad Hoc versus TGL Models

TGL provides models in a uniform manner to all its theories. Hence, a calculus specified in classic TGL receives a complete semantics. On the other hand, specific models have been proposed in the literature for various λ-calculi.

Here we compare some standard loose, set-theoretic models of the λ-calculus [7] and System F [8] with the default TGL models of their specifications. For the λ-calculus, the TGL semantics coincides (up to a carrier-preserving bijection between classes of models) with its ad hoc semantics. For System F, TGL provides a novel semantics, that we prove equivalent to the ad hoc one.

The fact that, in these two cases, the uniform TGL semantics achieves essentially the same effect as some carefully chosen ad hoc semantics, suggests TGL as a potential semantic framework for λ-calculi.[7]

5.1. Untyped λ-Calculus

We recall standard model-theoretic notions for the untyped λ-calculus described in [7] following an approach originally developed in [23]. Our discussion will cover both the extensional and the non-extensional variants.

[7] Note that TGL does not have native support for partiality—whereas all of our specified examples are variants of small-step semantics and hence are essentially total, other systems such as big-step λ-calculus could be better specified in a suitable "partial" variation of TGL; developing such a variation requires some careful design choices and is left for future work.

Let us call *pre-structure* a triple $\mathcal{A} = (A, _\langle_\rangle, (A_X)_{X \in Term(A)})$, where A is a set, $_\langle_\rangle$ is a binary operation on A (i.e., $(A, _\langle_\rangle)$ is an applicative structure), and $A_X : A^{Var} \to A$ for each $X \in Term(A)$, where $Term(A)$ is the set of λ-terms with constants in A. Given an equation $X_1 = X_2$ with X_1, X_2 λ-terms, $\mathcal{A} \models_\lambda X_1 = X_2$ is defined as $A_{X_1}(\rho) = A_{X_2}(\rho)$ for all $\rho : Var \to A$. For pre-structures, we consider the following properties (where a, b range over elements of A, x over variables, X, X_1, X_2 over terms, ρ, ρ' over valuations in A^{Var}):

(P1) $A_x(\rho) = \rho(x)$;

(P2) $A_{X_1X_2}(\rho) = A_{X_1}(\rho)\langle A_{X_2}(\rho)\rangle$;

(P3) If $\rho\restriction_{FV(X)} = \rho'\restriction_{FV(X)}$, then $A_X(\rho) = A_X(\rho')$;

(P4) If $A_X(\rho[x \leftarrow a]) = A_{X'}(\rho[x \leftarrow a])$ for all $a \in A$, then $A_{\lambda x.X}(\rho) = A_{\lambda x.X'}(\rho)$;

(P5) $A_{\lambda x.X}(\rho)\langle a\rangle = A_X(\rho[x \leftarrow a])$;

(P6) If $a\langle c\rangle = b\langle c\rangle$ for all $c \in A$, then $a = b$;

(P7) $A_a(\rho) = a$.

We next simplify the pre-structures slightly, by removing their redundant data given by parameterized terms. A *simple pre-structure* is a triple of the form $(A, _\langle_\rangle, (A_X)_{X \in Term})$; thus, unlike pre-structures which interpret extended terms from $Term(A)$, simple pre-structures interpret terms from $Term$, the set of pure λ-terms. The notion of satisfaction for simple pre-structures is defined in the same way as for pre-structures. We are only interested in pre-structures verifying at least (P1)-(P4). In this case, simple pre-structures and pre-structures are essentially identical:

Lemma 5.1. *The forgetful function sending* $(A, _\langle_\rangle, (A_X)_{X \in Term(A)})$ *to* $(A, _\langle_\rangle, (A_X)_{X \in Term})$ *is a bijection, preserving satisfaction and the properties (P5) and (P6), between pre-structures verifying (P1)-(P4) and (P7) and simple pre-structures verifying (P1)-(P4).*

This lemma allows us to switch to simple pre-structures, which we henceforth call "pre-structures", forgetting about the original notion, as well as about (P7).

A *syntactical λ-model* [7, Sect. 5.3] (*λ-model* for short) is a pre-structure verifying (P1)-(P5). A λ-model is *extensional* if it verifies (P6).[8]

[8]The concept of pre-structure does not exit in [7], where syntactical λ-models and their extensional variants are introduced directly as tuples $(A, _\langle_\rangle, (A_X)_{X \in Term(A)})$ satisfying the properties [(P1)-(P5), (P7)] and (P1)-(P7), respectively—more precisely, using the notations from [7, Def. 5.3.1], (P1) is (1), (P2) is (3), (P5) is (4), (P3) is (5) and (P7) is (2); moreover, (P4) is the semantic (ξ) condition from [7, Def. 5.3.2]. Thanks to Lemma 5.1, we can instead equivalently speak of tuples $(A, _\langle_\rangle, (A_X)_{X \in Term})$ (which we currently call "pre-structures") satisfying the properties (P1)-(P5) and (P1)-(P6), respectively.

The TGL representation $U\lambda_=$ from Subsection 4.4 employs an unsorted language over the syntax of λ-calculus, with no relation symbols. In this language, models have therefore the form $(A, (A_X)_{X \in Term})$.

We consider the following TGL formulas and formula-schemas:

$(\lambda x.X)X' = X[X'/x]$	(β)
$(\lambda x.X)x = X$	(β')
$\lambda x.X\, x = X,$ if $x \notin FV(X)$	(η)
$\lambda x.y\, x = y$	(η')
$(\forall x.X_1 = X_2) \Rightarrow \lambda x.X_1 = \lambda x.X_2$	(ξ)
$(\forall x.X_1\, x = X_2\, x) \Rightarrow X_1 = X_2,$ if $x \notin FV(X_1, X_2)$	(ext)
$(\forall x.y_1 x = y_2 x) \Rightarrow y_1 = y_2$	(ext$'$)

Lemma 5.2. *Each of* (β), (η), (ext) *is equivalent in TGL to its primed variant.*

This lemma illustrates once more the TGL mechanism of eliminating low-level details: (β) can be compactly expressed as the substitution-free (β'), and (η) and (ext) as the side-condition-free (η') and (ext$'$), respectively.

A main difference between pre-structures and TGL models is that the former additionally provide an application operator—this can bee recovered from the TGL interpretation of syntactic application. We define a correspondence between pre-structures verifying (P1)-(P4) and TGL models satisfying (ξ):
- Each pre-structure $\mathcal{A} = (A, _\langle_\rangle, (A_X)_{X \in Term})$ verifying (P1)-(P4) is mapped to a TGL model $\mathcal{A}^{\#} = (A, (A_X)_{X \in Term})$;
- Each TGL model $\mathcal{A} = (A, (A_X)_{X \in Term})$ satisfying (ξ) is mapped to a pre-structure $\mathcal{A}^{\$} = (A, _\langle_\rangle, (A_X)_{X \in Term})$, where $_\langle_\rangle$ is defined by $a\langle b\rangle = A_{xy}(\rho)$, with ρ taking x to a and y to b.

Proposition 5.3. *The above two mappings are mutually inverse. Moreover, they preserve satisfaction and can be restricted and corestricted to:*
(1) λ-models versus TGL models of $(\xi), (\beta)$ (models of $U\lambda_=$ without (η));
(2) extensional λ-models versus TGL models of $(\xi), (\beta), (\eta)$ (models of $U\lambda_=$).

In other words, λ-models and extensional λ-models coincide with the models of the corresponding TGL theories.

Adequacy is an immediate consequence of Proposition 5.3 together with the soundness and completeness of λ-calculus deduction w.r.t. the syntactic models (see [7, Theorem 5.3.4] and [7, Theorem 5.2.18] via [7, Theorem 5.3.6]). We write $\vdash_\beta$ for deduction in $\beta\lambda$-calculus and $\vdash_{\beta\eta}$ for deduction in $\beta\eta\lambda$-calculus.

Corollary 5.4. *The following hold for all terms X, X':*
(1) $\vdash_\beta X = X'$ iff $(\xi), (\beta) \vdash_{TGL} X = X'$.
(2) $\vdash_{\beta\eta} X = X'$ iff $U\lambda_= \vdash_{TGL} X = X'$.

5.2. System F

Recall from Subsection 4.1 that the term syntax for System F has two sorts, *data* and *type*, that Var_{type} (ranged over by t) and $Term_{type}$ (ranged over by T) denote the sets of variables and terms of sort *type*, and similarly for *data* variables and terms, ranged over by x and X, respectively.

Since typing is not affected by equational reasoning, a version of Proposition 4.1 from Subsection 4.2 holds for $\mathcal{SF}_=$ instead of $\mathcal{TSF}$:

Proposition 5.5. *The following are equivalent:*
(a) $\vdash_{TSF} \Gamma \triangleright X : T$.
(b) $\mathcal{SF}_= \vdash_{TGL} \Gamma^{\#} \triangleright tpOf(X, T)$.

Next we recall Henkin models, a standard notion of models for System F introduced by Bruce, Meyer and Mitchell [8]— they were proved complete w.r.t. the equational theory of System F. Such a model will be defined below to consist of various components, among which a set $\mathcal{T}$ for interpreting type terms and a family of sets $(\mathcal{D}om_\tau)_{\tau\in\mathcal{T}}$ for interpreting data terms. A type valuation γ will be an element of $\mathcal{T}^{Var_{type}}$, i.e., a function from Var_{type} to $\mathcal{T}$. The model will also provide a family of functions for interpreting type terms relative to type valuations, $(H_T)_{T\in Term_{type}}$, with $H_T : \mathcal{T}^{Var_{type}} \to \mathcal{T}$. A data valuation δ will be a function from Var_{data} to $\mathcal{D}om$. The pair (γ, δ) is said to be compatible with a typing context Γ if $\delta(x) \in H_T(\gamma)$ for all $x{:}T \in \Gamma$. We write $Compat(\Gamma)$ for the set of such pairs. In addition, for each typing judgment $tj = (\Gamma \triangleright X : T)$ and Γ-compatible valuations (γ, δ), a model will provide an interpretation $H_{tj}(\gamma, \delta) \in \mathcal{D}om_{H_T(\gamma)}$. We let Tj be the set of typing judgments.

Definition 5.6. *[8, 29] A* Henkin model $\mathcal{H}$ *for System F is a tuple* $(\mathcal{T}, \mathcal{F}, \to, \Pi, (\mathcal{D}om_\tau)_{\tau\in\mathcal{T}}, (App_{\tau,\sigma})_{\tau,\sigma\in\mathcal{T}}, (App_f)_{f\in\mathcal{F}},)$*, together with a pair* $((H_T)_{T\in Term_{type}}, (H_{tj})_{tj\in Tj, \vdash_{SF} tj})$*, where:*
(a) $\to : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$*,*
(b) $\Pi : \mathcal{F} \to \mathcal{T}$*,*
(c) $\mathcal{F} \subseteq \mathcal{T}^{\mathcal{T}}$*,*
(d) $App_{\tau,\sigma} : \mathcal{D}om_{\tau\to\sigma} \to \mathcal{D}om_\sigma{}^{\mathcal{D}om_\tau}$ *for each* $\tau, \sigma \in \mathcal{T}$*,*
(e) $App_f : \mathcal{D}om_{\Pi f} \to \prod_{\tau\in\mathcal{T}} \mathcal{D}om_{f(\tau)}$ *for each* $f \in \mathcal{F}$*,*
(f) $H_T : \mathcal{T}^{Var_{type}} \to \mathcal{T}$ *for each* $T \in Term_{type}$*,*
(g) $H_{\Gamma\triangleright X:T} : \prod_{(\gamma,\alpha)\in Compat(\Gamma)} \mathcal{D}om_{H_T(\gamma)}$*,*
such that the following hold:
(1) $App_{\tau,\sigma}$ *and* App_f *are injective;*
(2) $(\tau \mapsto H_T(\gamma[t \leftarrow \tau])) \in \mathcal{F}$ *for each* T, t, γ*;*
(3) $H_t(\gamma) = \gamma(t)$ *for each* $t \in Var_{type}$*;*
(4) $H_{T\to T'}(\gamma) = H_T(\gamma) \to H_{T'}(\gamma)$*;*
(5) $H_{\Pi t.T}(\gamma) = \Pi(\tau \mapsto H_T(\gamma[t \leftarrow \tau]))$*;*
(6) $H_{\Gamma\triangleright x:T}(\gamma, \delta) = \delta(x)$*;*
(7) $H_{\Gamma\triangleright XY:T}(\gamma, \delta) = App_{H_{T'}(\gamma),H_T(\gamma)}(H_{\Gamma\triangleright X:T'\to T}(\gamma, \delta))(H_{\Gamma\triangleright Y:T'}(\gamma, \delta))$*;*
(8) $H_{\Gamma\triangleright XT:T'[T/t]}(\gamma, \delta) = App_{\tau\mapsto H_{T'}(\gamma[t\leftarrow\tau])}(H_{\Gamma\triangleright X:\Pi t.T'}(\gamma, \delta))(H_T(\gamma))$*;*
(9) $H_{\Gamma\triangleright\lambda x:T.X:T\to T'}(\gamma, \delta) \in \mathcal{D}om_{H_T(\gamma)\to H_{T'}(\gamma)}$ *and, for each* $d \in \mathcal{D}om_{H_T(\gamma)}$*,*

$App_{H_T(\gamma),H_{T'}(\gamma)}(H_{\Gamma \triangleright \lambda x:T.X:T \to T'}(\gamma,\delta))(d) = H_{\Gamma,x:T \triangleright X:T'}(\gamma,\delta[x \leftarrow d])$;
(10) $H_{\Gamma \triangleright \lambda t.X:\Pi t.T}(\gamma,\delta) \in \mathcal{D}om_{\Pi(\tau \mapsto H_T(\gamma[t \leftarrow \tau]))}$ *and, for each* $\tau \in \mathcal{T}$,
$App_{\tau \mapsto H_T(\gamma[t \leftarrow \tau])}\left(H_{\Gamma \triangleright \lambda t.X:\Pi t.T}(\gamma,\delta)\right)(\tau) = H_{\Gamma \triangleright X:T}(\gamma[t \leftarrow \tau],\delta)$.

Above, we used slightly different notations than [29]; also, we included the interpretation functions H_T and H_{tj} as part of the structure, while [29] equivalently asks that such interpretations exist and then proves them to be unique.

Satisfaction by Henkin models $\mathcal{H}$ of well-typed equations $\Gamma \triangleright X = Y : T$ (with $\vdash_{TSF} \Gamma \triangleright X : T$ and $\vdash_{TSF} \Gamma \triangleright X : T$) is defined by $H \models_{SF} \Gamma \triangleright X = Y : T$ iff $H_{\Gamma \triangleright X:T} = H_{\Gamma \triangleright Y:T}$.

We now establish the relationship between Henkin models and TGL models satisfying $\mathcal{SF}_{=}$. For a TGL model $\mathcal{M}$, we let M_{type} and M_{data} be the elements of sort *type* and *data*, respectively. (Thus, the carrier set M is the disjoint union of M_{type} and M_{data}.) A (well-sorted) valuation in such a model can therefore be identified with a pair $(\gamma : Var_{type} \to M_{type}, \delta : Var_{data} \to M_{data})$. Moreover, since type terms T do not contain data variables, by Lemma 2.5 we have that $M_T(\gamma,\delta)$ does not depend of δ, hence we simply write $M_T(\gamma)$ for it.

Unlike TGL models, Henkin models provide explicit semantic counterparts for some term-syntax operators: $\to$, Π, $App_{\tau,\sigma}$ and App_f. However, as shown in Subsection 5.1 for the untyped λ-calculus, this is not a fundamental difference, since these operations arise naturally from a TGL interpretation. E.g., in a TGL model $\mathcal{M}$, a first-order operator such as $App_{\tau,\sigma}$ can be defined by $App_{\tau,\sigma}(d)(d') = M_{x\,y}(\gamma,\delta)$ for any choice of γ and δ where γ maps x to d and y to d'. To define second-order operators such as Π, we first define a type-polynomial to be a function $f : M_{type} \to M_{type}$ of the form $\tau \mapsto M_T(\gamma[t \leftarrow \tau])$ for some T, t, γ. For type-polynomials f, $\Pi(f)$ is taken to be $M_{\Pi\, t.\, T}(\gamma)$, where T, t, γ are as above—their choice is immaterial due to the axiom $(\mathsf{T}\text{-}\xi)$ of $\mathcal{SF}_{=}$. (Only the definition of Π on type-polynomials is relevant w.r.t. Henkin semantics.)

A second difference is that TGL models interpret data terms, while Henkin models interpret judgments. This can be dealt with noticing that, in Henkin models, Γ and T from $\Gamma \triangleright X : T$ are only aimed at directing the interpretation of X w.r.t. typing by offering types to its free variables and a result type. Therefore, given a Henkin model $\mathcal{H}$ and a data term X, $H_X(\delta,\gamma)$ can be defined as $H_{\Gamma \triangleright X:T}(\gamma,\delta)$ if X is typable, and Γ and T are such that $\vdash_{SF} \Gamma \triangleright X : T$ and (γ,δ) is compatible with Γ. For non-well-typed terms, $H_X(\delta,\gamma)$ is irrelevant, and we can map it to some "error element" *err*.

Finally, a third (minor) difference is that Henkin models classify data into types through a family $(\mathcal{D}om_\tau)_{\tau \in \mathcal{T}}$, while TGL models employ a binary relation M_{tpOf} between data and types—these views are of course equivalent.

These lead us to define the following transformations:
Henkin to TGL: For each Henkin model $\mathcal{H}$, we define a TGL model $\mathcal{M} = \mathcal{H}^{\#}$:
(a) $M_{type} = \mathcal{T}$;
(b) $M_{data} = \bigcup_{\tau \in \mathcal{T}} \mathcal{D}om_\tau \cup \{err\}$, where $err \notin \bigcup_{\tau \in \mathcal{T}} \mathcal{D}om_\tau$;
(c) $M_T(\gamma) = H_T(\gamma)$;
(d) $M_X(\gamma,\delta) = H_{\Gamma \triangleright X:T}(\gamma,\delta)$ if there exist Γ and T such that $\vdash_{SF} \Gamma \triangleright X : T$ and (γ,δ) is compatible with Γ in $\mathcal{H}$; otherwise $M_X(\gamma,\delta) = err$;

(e) $(d, \tau) \in M_{tpOf}$ iff $d \in \mathcal{D}om_\tau$.

TGL to Henkin: For each TGL model $\mathcal{M}$ satisfying $\mathcal{SF}_=$, we define a Henkin model $\mathcal{H} = \mathcal{M}^\$$:
(a) $\mathcal{T} = M_{type}$;
(b) $\mathcal{F} = \{f : \mathcal{T} \to \mathcal{T}.\ f$ type-polynomial in $\mathcal{M}\}$;
(c) $\tau \to \sigma = M_{t \to t'}(\gamma)$, where $\gamma(t) = \tau$ and $\gamma(t') = \sigma$;
(d) $\Pi(f) = M_{\Pi t.T}(\gamma, \delta)$, if f is $\tau \mapsto M_T(\gamma[t \leftarrow \tau], \delta)$;
(e) $\mathcal{D}om_\tau = \{d \in M_{data}.\ (d, \tau) \in M_{tpOf}\}$;
(f) $App_{\tau,\sigma}(d)(d') = M_{x\,y}(\gamma, \delta)$, where $\delta(x) = d$, $\delta(y) = d'$;
(g) $App_f(d)(\tau) = M_{x\,t}(\gamma, \delta)$, where $\gamma(t) = \tau$ and $\delta(x) = d$;
(h) $H_T(\gamma) = M_T(\gamma)$;
(i) $H_{\Gamma \triangleright X:T}(\gamma, \delta) = M_X(\gamma, \delta)$.

According to the above discussion, these transformations between TGL and Henkin models can be seen as inessential variations, obtained by slightly reorganizing the components. In particular, the transformations preserve and reflect the satisfaction of System F equational judgments:

Proposition 5.7. *Assume* $\vdash_{SF} \Gamma \triangleright X:T$ *and* $\vdash_{SF} \Gamma \triangleright Y:T$. *Then:*
(1) $\mathcal{H} \models_{SF} \Gamma \triangleright X = Y : T$ *iff* $\mathcal{H}^\# \models_{TGL} \Gamma^\# \triangleright X = Y$;
(2) $\mathcal{M} \models_{TGL} \Gamma^\# \triangleright X = Y$ *iff* $\mathcal{M}^\$ \models_{SF} \Gamma \triangleright X = Y : T$.

Using the completeness theorem from [8], we obtain the following:

Corollary 5.8. *The TGL theory* $\mathcal{SF}_=$ *is adequate for both the typing system and the equational theory of System F.*

6. Term-Generic Logic and HOAS

As we have seen, the specification of calculi and type systems as TGL theories provides a higher-level notation, in particular, it removes the need for freshness side-conditions and explicit substitution. Moreover, typing contexts are captured implicitly by the TGL contexts. These would qualify the TGL specification style as a variant of higher-order abstract syntax (HOAS) [33, 21, 27, 32, 15, 12].

A main difference from HOAS is that TGL uses the original syntax of a given system (provided it constitutes a term syntax, i.e., has well-behaved substitution and free-variables operators), while HOAS encodes the system's syntax in the syntax of a different logic, typically, higher-order logic or dependent type theory. Moreover, TGL is a classic logic, with set-theoretic models giving direct semantics to the term syntax, while HOAS typically employs intuitionistic reasoning mechanisms and is not concerned with models. The generic relationship between term syntax and semantics (studied in Section 3 and instantiated in Section 5) is an aspect covered by TGL, but not by HOAS.

6.1. Additional flexibility of TGL compared to HOAS

HOAS is very well-suited, and greatly simplifies much of the representation and proof bureaucracy, in case the object system has similar binding mechanisms with the logical framework. If this is not the case though, a HOAS representation can be impossible or at best challenging. Examples include non-standard λ-calculi and general binders. However, the abstract TGL term-syntax axioms still apply in these cases, for the natural notions of substitution and free variables.

Illative and linear λ-calculi [7, 18, 26] are versions of λ-calculi with λ-bound terms restricted to respectively contain the bound variable (in the case of the illative λ-calculi) and contain precisely one occurrence of the bound variable (in the case of the linear λ-calculi). The grammar for terms is $X ::= x \mid X\,Y \mid \lambda x.Y$, just like for standard λ-calculus, except that for the production $X ::= \lambda x.Y$ one additionally requires:
- that $x \in FV(Y)$, for the illative case;
- that x appears free in Y precisely once, for the linear case.

Since the illative and linear terms are closed under substitution and α-equivalence we obtain that the TGL term-syntax axioms hold for (α-equivalence classes of) these terms.

General binders in the style of Ott [45] and the latest Nominal Isabelle [51] allow for binding in terms not only single variables, but also arbitrarily complex (perhaps recursive) structures containing variables: one can bind simultaneously a whole list, or a finite set of variables, or even a tree structure. As a simple example, $\mathsf{let}\ (x_1, x_2) = (X_1, X_2)\ \mathsf{in}\ Y$ binds simultaneously x_1 and x_2 in Y. In general one has, besides the syntactic categories of terms and variables, a category Binder of binders, perhaps defined mutually recursively with terms; in particular, binders can occur in terms, having clearly indicated scopes. There is also a notion of the set or list of variables that a binder binds, $\mathsf{binds} : \mathsf{Binder} \to K(\mathit{Var})$, where K is either $\mathcal{P}_f$ or List. Free variables and substitution of terms are defined as one would expect, the latter avoiding capture by $\mathsf{binds}(b)$ in the scope of b when a binder b is encountered during the recursive traversal of a term. α-equivalence is defined on raw terms using binds, with or without neglecting vacuous binders (depending on the selected "mode"). It is fairly easy to check that, even in this general case, terms modulo α form a TGL term syntax.

6.2. Limitations of TGL and the prospect of combining it with HOAS

Unlike in HOAS, most interesting TGL theories for calculi with bindings employ axiom-schemas, and consequently are not directly suitable for mechanical reasoning (in a theorem prover). The culprit for this is TGL's underspecification/parametrization: it does not provide a means to represent syntax of terms, but is parameterized by such a syntax.

This restriction can be addressed by implementing (instances of) TGL in a framework that does provide syntax specification mechanisms, including higher-order support for axiom-schemas. For example, both the Edinburgh LF [21] and the Isabelle [32] frameworks are based on HOAS and are therefore specialized in representing syntax with bindings. For such a framework, the schemas of the

TGL theories are no more problematic than the schemas residing at the core of TGL, and of FOL for that matter. Therefore, *if a term syntax is representable in such a framework, then a TGL specification over this syntax will also be representable.*

To make the above claim more concrete, let S be a finite set of sorts and $\Sigma = (\Sigma_{w,v})_{w \in S^*, v \in (S \times S)^*}$ a finite *binding signature*, where $f \in \Sigma_{w,v}$, with $w = s_1 \dots s_m$ and $v = ((t_1, t'_1), \dots, (t_n, t'_n))$, indicates that f takes m free arguments of sorts s_i and n bound arguments, binding variables of sort t_j in terms of sort t'_j. Then the S-sorted set $Term_\Sigma$, of terms up to α-equivalence, forms a term syntax—this covers our examples from Section 4. Moreover, if we consider a finite S^*-ranked set of relation symbols Π, then the language $(Term_\Sigma, \Pi)$ is representable in LF or Isabelle as follows:
- The term syntax is represented in the usual fashion: each sort s is declared as a type $\underline{s}$ and each function symbol $f \in \Sigma_{w,v}$, with $w = s_1 \dots s_m$ and $v = ((t_1, t'_1), \dots, (t_n, t'_n))$, is declared as a constant $\underline{f} : \underline{s_1} \to \dots \to \underline{s_m} \to \dots (\underline{t_1} \to \underline{t'_1}) \to \dots \to (\underline{t_n} \to \underline{t'_n})$.
- The TGL formulas on top of this syntax are represented in the same way as those of FOL: each $\pi \in \Pi_w$ with $w = s_1 \dots s_n$ becomes $\underline{\pi} : \underline{s_1} \to \dots \to \underline{s_n} \to o$ in LF (where o is a newly declared type of formulas); for each sort s, the s-universal quantifier is represented by a constant $\mathsf{All}^s : (s \to o) \to o$; etc.

Finally, the TGL reasoning infrastructure can be represented in the same way as that of FOL, since it does not depend on the concrete format of terms; representations of sequent-style and natural-deduction style for FOL are well-known in both LF and Isabelle [21, 4, 31]. In LF, one defines a new dependent type family $\mathsf{true} : o \to \mathsf{Type}$ representing the "truth" judgment for formulas; $\mathsf{true}\ \varphi$ will be inhabited by all the proofs of φ.

Let us illustrate the similarity between the TGL term schematism and the FOL formula schematism by an example. First consider standard FOL over one single sort, with function symbols for zero and successor. These are represented in LF by declaring a type ι and constants $0 : \iota$ and $\mathsf{Suc} : \iota \to \iota$. If we want to represent an axiom schema such as the following induction principle, schematic in the formula φ and its free variable x:

$$\varphi(0/x) \wedge (\forall x.\, \varphi \Rightarrow \varphi[\mathsf{Suc}(x)/x]) \;\Rightarrow\; \forall x.\, \varphi \quad (\mathsf{Ind})$$

we would write something like this (representing an LF constant declaration):

$$\mathsf{Ind} : \{\varphi : \iota \to o\}\ \mathsf{true}\,(\varphi\ 0 \wedge (\mathsf{All}\ x.\ \varphi\ x \Rightarrow \varphi\,(\mathsf{Suc}\ x)) \Rightarrow \mathsf{All}\ x.\ \varphi\, x)$$

The same approach can be used to represent the TGL axiom-schema for our calculi specifications, just that this time the schematism applies not to formulas, but to terms. For example, the TGL schema for (ξ), namely $(\forall x.\ X = Y) \Rightarrow \lambda x.X = \lambda x.Y$, can be represented as

$$\xi : \{X, Y : \iota \to \iota\}\ \mathsf{true}\,((\mathsf{All}\ x.\ X\ x = Y\ x) \Rightarrow (\mathsf{Lam}\ x.\, X\ x) = (\mathsf{Lam}\ x.\, Y\ x))$$

Above, the braces { } indicate LF products, i.e., meta-level universal quantifiers. To ease readability, we used the following notations:

- the LF constants representing object-level conjunction, implication, and equality, $\wedge, \Rightarrow : o \to o \to o$ and $= : \iota \to \iota \to o$, are applied in an infix manner;
- for the LF constant representing object-level $\forall$-quantification, $\mathsf{All} : (\iota \to o) \to o$, and for any LF term $\varphi : \iota \to o$, we write $\mathsf{All}\ x.\, \varphi\ x$ instead of $\mathsf{All}(\lambda x.\ \varphi)$; and similarly for the LF constant $\mathsf{Lam} : (\iota \to \iota) \to \iota$ representing object-level λ-abstraction.

Given a TGL theory *with HOAS-representable term syntax*, and considering a representation of its schemas as exemplified above, we can prove the standard adequacy theorems [21]: one for terms and one for the TGL deduction. A potential future application of marrying HOAS with TGL is a framework for obtaining HOAS adequacy for free: Since establishing the adequacy of a HOAS representation of TGL (including TGL axiom-schemas of a given format) is a one-time effort, this indirection through TGL could be used to automate adequacy proofs. A similar goal is being pursued by λ-free frameworks [2] and canonical adaptations of LF [22].

7. Concluding Remarks

In this paper, we made the following contributions:

- We introduced *term-generic first-order logic (TGL)*, a logic where terms are "black-boxes" exporting substitution and free variables. Defining TGL models and proving completeness of a FOL-like Genzten system, we showed that the development of first-order logic up to the completeness theorem does not depend on the syntax for terms.

- We showed how various calculi and type systems can be defined as theories in TGL. For two of these calculi, we showed that the general-purpose TGL semantics is equivalent with their standard ad-hoc semantics. Compared to ad-hoc semantics, TGL semantics has the advantage of uniformity. For a new calculus specified as a TGL theory, one does not need to redesign a notion of model and prove a completeness theorem, but can use the default TGL models "customized" by the axioms of the theory.

- We discussed the relationship between TGL and HOAS, including a possible representation of TGL term syntax and deduction in HOAS

While the idea of developing first-order logic on top of an abstract term syntax that only exports substitution and free variables seems new, the literature abounds in approaches to represent syntax with bindings and λ-calculi.

First, Hindley and Longo [23, Remark 3.3] sketch a presentation of λ-calculus models as extensions of FOL models to account for λ-terms (and in particular propose the same axioms for (η) and (ξ) as we did in Subsection 4.4). Continuations and generalizations of their approach can be tracked in the notions of binding algebra [1, 48] and binding logic [11]. In these works, terms have a concrete structure, with constructors indicated by binding signatures—they can be shown to form TGL term syntaxes. On the semantic side, the models

from these settings and our TGL models display an evolution towards looseness. Specifically, binding algebras are extensional structures, where bindings are mapped to second-order functions. In order to also distinguish intensional structure if needed, binding logic introduces more relaxed models, where bound terms are interpreted as abstract items endowed with abstract application operators, which may or may not be extensional. Finally, TGL models are not even required to interpret bound terms as "applicable" items, be they extensional or intensional; instead, the interpretation is only required to compose well with substitution. As shown in Section 5, the functional interpretation, or a more intensional behavior of application, can be recovered by TGL axioms—this could be used to prove that the aforementioned logics are embeddable in TGL, both syntactically and semantically.

In Section 6, we have already discussed higher-order abstract syntax (HOAS), a methodology that delegates variables and bindings of the represented system to the meta-language variables and bindings. By contrast, first-order approaches such as de Bruijn levels [9], de Bruijn indexes [9], nominal logic [35, 50], locally nameless [19, 20, 12, 6] or locally named [44, 36], represent variables as objects that can be referred explicitly in the logic. The considered terms usually have static bindings and capture-avoiding substitution, therefore constituting TGL term syntaxes. While most of these approaches develop purely syntactic methods, work on nominal logic also covers semantics. Faithful to the explicit-variable view, nominal models/algebras [14, 10] are inhabited by abstract syntactic objects representing names (variables). By contrast, in TGL variables have no semantic meaning in the absence of a valuation. This makes it difficult to represent cases where the object-calculus unbound variables have their own "individuality" distinct from that given by the terms that will substitute them [5]—this limitation, as well as some of its workarounds [4, 34], seems to be shared by TGL with HOAS.

Our own previous work [37, 39, 38] combines HOAS with a first-order approach in a definitional framework implemented in Isabelle/HOL [30]: after an instrumentation of substitution in a first-order setting, HOAS machinery is added as a definitional layer. By contrast, TGL (as well as its prototyped Isabelle implementation) argues for axiomatic specifications of systems. The basic term syntax components—free/fresh variables and substitution—are involved in a characterization of terms with bindings as initial model in a category of algebras, yielding substitution-compositional recursion principles [38].

Inspired by the method of de Bruijn levels, functor-category approaches [13, 24, 3] organize the terms with bindings as a presheaf indexed by the sets of free variables. While TGL models are based on different principles than presheaf models, TGL term syntaxes export enough structure and properties to form presheaf models, since the latter only require free variables and well-behaved renaming (a particular case of substitution).

References

[1] P. Aczel. Frege structures and the notions of proposition, truth and set. In *The Kleene Symposium*, pages 31–59. North Holland, 1980.

[2] R. Adams. *A modular hierarchy of logical frameworks*. PhD thesis, University of Manchaster, 2004.

[3] S. J. Ambler, R. L. Crole, and A. Momigliano. A definitional approach to primitive recursion over higher order abstract syntax. In *MERLIN*, 2003.

[4] A. Avron, F. Honsell, I. A. Mason, and R. Pollack. Using typed λ-calculus to implement formal systems on a machine. *Journal of Automatic Reasoning*, 9(3):309–354, 1992.

[5] B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanized metatheory for the masses: The poplmark challenge. In *TPHOLs*, pages 50–65, 2005.

[6] B. E. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *POPL*, pages 3–15, 2008.

[7] H. P. Barendregt. *The Lambda Calculus*. North-Holland, 1984.

[8] K. B. Bruce, A. R. Meyer, and J. C. Mitchell. The semantics of second-order lambda calculus. *Information and Computation*, 85(1):76–134, 1990.

[9] N. Bruijn. λ-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Math*, 34(5):381–392, 1972.

[10] J. Cheney. Completeness and Herbrand theorems for nominal logic. *Journal of Symbolic Logic*, 71(1):299–320, 2006.

[11] G. Dowek, T. Hardin, and C. Kirchner. Binding logic: Proofs and models. In *LPAR*, volume 2514, pages 130–144, 2002.

[12] A. P. Felty and A. Momigliano. Hybrid - a definitional two-level approach to reasoning with higher-order abstract syntax. *Journal of Automatic Reasoning*, 48(1):43–105, 2012.

[13] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *LICS Conf'99*, pages 193–202, 1999.

[14] M. J. Gabbay. Nominal algebra and the HSP theorem. *Journal of Logic and Computation*, 19(2):341–367, 2009.

[15] A. Gacek, D. Miller, and G. Nadathur. A two-level logic approach to reasoning about computations. *Journal of Automatic Reasoning*, 49(2):241–273, 2012.

[16] J. H. Gallier. *Logic for computer science. Foundations of automatic theorem proving*. Harper & Row, 1986.

[17] J.-Y. Girard. Une extension de l'interpretation de Gödel a l'analyse, et son application a l'elimination des coupure dans l'analyse et la theorie des types. In *2nd Scandinavian Logic Symposium*, pages 63–92. North Holland, 1971.

[18] J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[19] A. D. Gordon. A mechanisation of name-carrying syntax up to alpha-conversion. In *HUG*, pages 413–425, 1994.

[20] A. D. Gordon and T. F. Melham. Five axioms of alpha-conversion. In *TPHOLs*, pages 173–190, 1996.

[21] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *LICS'87*, pages 194–204, 1987.

[22] R. Harper and D. R. Licata. Mechanizing metatheory in a logical framework. *Journal of Functional Programming*, 17(4-5):613–673, 2007.

[23] J. R. Hindley and G. Longo. Lambda calculus models and extensionality. *Z. Math. Logik Grundlag Math.*, 29:289–310, 1980.

[24] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *LICS'99*, pages 204–213, 1999.

[25] J.-L. Krivine. *Lambda-calculus, types and models*. Ellis Horwood series in computers and their applications. Masson, 1993.

[26] P. Lincoln and J. C. Mitchell. Operational aspects of linear lambda calculus. In *LICS*, pages 235–246, 1992.

[27] R. C. McDowell and D. A. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, 2002.

[28] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts i and ii. *Information and Computation*, 100(1):1–77, 1992.

[29] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[30] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer, 2002.

[31] L. C. Paulson. Isabelle's logics: overview and misc logics. `http://www.cl.cam.ac.uk/research/hvg/Isabelle/documentation.html`.

[32] L. C. Paulson. The foundation of a generic theorem prover. *Journal of Automatic Reasoning*, 5(3):363–397, 1989.

[33] F. Pfenning and C. Elliot. Higher-order abstract syntax. In *PLDI '88*, pages 199–208, 1988.

[34] B. Pientka. Proof pearl: The power of higher-order encodings in the logical framework lf. In *TPHOLs*, pages 246–261, 2007.

[35] A. M. Pitts. Nominal logic: A first order theory of names and binding. In *TACS'01*, pages 219–242, 2001.

[36] R. Pollack, M. Sato, and W. Ricciotti. A canonical locally named representation of binding. *Journal of Automatic Reasoning*, 49(2):185–207, 2012.

[37] A. Popescu. Contributions to the theory of syntax with bindings and to process algebra. Ph.D. Thesis, Univ. of Illinois, 2010. Available at `http://hdl.handle.net/2142/18477`.

[38] A. Popescu and E. L. Gunter. Recursion principles for syntax with bindings and substitution. In *ICFP*, pages 346–358, 2011.

[39] A. Popescu, E. L. Gunter, and C. J. Osborn. Strong normalization for System F by HOAS on top of FOAS. In *LICS*, pages 31–40, 2010.

[40] A. Popescu and G. Roşu. Term-generic logic. Extended Technical Report. `http://www21.in.tum.de/~popescua/TGL-report.pdf`.

[41] A. Popescu and G. Rosu. Term-generic logic. In *WADT*, pages 290–307, 2008.

[42] J. C. Reynolds. Towards a theory of type structure. In *Symposium on Programming*, pages 408–423, 1974.

[43] J. C. Reynolds. Types, Abstraction and Parametric Polymorphism. In *IFIP Congress*, pages 513–523, 1983.

[44] M. Sato and R. Pollack. External and internal syntax of the lambda-calculus. *Journal of Symbolic Computation*, 45:598–616, 2010.

[45] P. Sewell, F. Z. Nardelli, S. Owens, G. Peskine, T. Ridge, S. Sarkar, and R. Strnisa. Ott: Effective tool support for the working semanticist. *J. Funct. Program.*, 20(1):71–122, 2010.

[46] R. Statman. Logical relations and the typed lambda-calculus. *Information and Control*, 65(2/3):85–97, 1985.

[47] A. Stoughton. Substitution revisited. *Theor. Comput. Sci.*, 59:317–325, 1988.

[48] Y. Sun. An algebraic generalization of Frege structures - binding algebras. *Theoretical Computer Science*, 211(1-2):189–232, 1999.

[49] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996.

[50] C. Urban. Nominal techniques in Isabelle/HOL. *Journal of Automatic Reasoning*, 40(4):327–356, 2008.

[51] C. Urban and C. Kaliszyk. General bindings and alpha-equivalence in nominal isabelle. *Logical Methods in Computer Science*, 8(2), 2012.

[52] P. Wadler. Theorems for free! In *FPCA '89*, pages 347–359. ACM, 1989.

APPENDIX

A. Definitions of Operators on Formulas

Below we list these (very standardly-looking) definitions. The set $FV(\varphi)$, of the *free variables* of φ, is defined recursively as follows: $FV(\pi(T_1,\ldots,T_n)) = FV(T_1)\cup\ldots\cup FV(T_n)$; $FV(\varphi\Rightarrow\psi) = FV(\varphi)\cup FV(\psi)$; $FV(\varphi\wedge\psi) = FV(\varphi)\cup FV(\psi)$; $FV(\forall x.\varphi) = FV(\varphi)\setminus\{x\}$. (Note that, since $FV(T)$ is finite for each term T, $FV(\varphi)$ is also finite for each formula φ.) The operator FV is extended from terms and formulas to sets, lists, and tuples of terms and/or formulas in the expected way, by taking the union.

The *(capture-free) substitution map* (of terms for variables in formulas) $Subst : Formula\times Term^{Var,sort}\to Formula$ is defined as follows: $Subst\,(\pi(T_1,\ldots,T_n),\theta) = \pi(Subst\,(T_1,\theta),\ldots,Subst\,(T_n,\theta))$; $Subst\,(\varphi\Rightarrow\psi,\theta) = Subst\,(\varphi,\theta)\Rightarrow Subst\,(\psi,\theta)$; $Subst\,(\varphi\wedge\psi,\theta) = Subst\,(\varphi,\theta)\wedge Subst\,(\psi,\theta)$; $Subst\,(\forall x.\varphi,\theta) = \forall z.Subst\,(\varphi,\theta[x\leftarrow z])$, where z is the least variable (for some fixed wellorder on variables) having sort $sort\,(x)$ and not in $FV(\varphi)\cup\bigcup\{FV(\theta(y)) : y\in FV(\varphi)\}$.

α-equivalence of formulas, written $\equiv_\alpha$, is defined to be the smallest relation $R\subseteq Formula\times Formula$ satisfying the following: $\pi(T_1,\ldots,T_n)\;R\;\pi(T_1,\ldots,T_n)$; $\varphi\wedge\varphi'\;R\;\psi\wedge\psi'$ if $\varphi\;R\;\psi$ and $\varphi'\;R\;\psi'$; $\varphi\Rightarrow\varphi'\;R\;\psi\Rightarrow\psi'$ if $\varphi\;R\;\psi$ and $\varphi'\;R\;\psi'$; $\forall x.\varphi\,R\,\forall y.\psi$ if $sort\,(x) = sort\,(y)$, $\varphi[z/x]\,R\,\psi[z/y]$ for some $z\notin FV(\varphi)\cup FV(\psi)$ with $sort\,(z) = sort\,(x)$.

B. Proofs of basic lemmas in TGL

Recall that $[T/x]$ denotes the function in $Var\to Term$ sending x to T and all other variables to themselves. Moreover, recall our convention: Whenever we write $K[T/x]$ (where K is a term or a formula), we implicitly assume that the substitution is well-defined, i.e., that $sort\,(T) = sort\,(x)$. We shall also write ι for $Var\hookrightarrow Term$, the inclusion of variables into terms.

Proof of Lemma 2.2. (1) Assume $x\notin FV(T)$. Since $[T'/x]\restriction_{FV(T)} = \iota\restriction_{FV(T)}$,

with Definition 2.1.(3,2) we obtain $T[T'/x] = \mathit{Subst}\,(T, \iota) = T$.
(2) If $y = x$ then, using Definition 2.1.(1), $y[T/x] = \mathit{Subst}\,(x, [T/x]) = [T/x](x) = T$. If $y \neq x$ then $[T/x]\restriction_{FV(y)} = \iota\restriction_{FV(y)}$, thus, with Definition 2.1.(3,2), $y[T/x] = \mathit{Subst}\,(y, \iota) = y$.
(3) $FV(T[T'/x]) = FV(\mathit{Subst}\,(T, [T'/x])) =$ (using Definition 2.1.(6))
$\bigcup\{FV([T'/x](y)).\ y \in FV(T)\} =$ (since $x \in FV(T)$)
$FV([T'/x](y)) \cup \bigcup\{FV([T'/x](y)) : y \in FV(T),\ y \neq x\} =$ (by point (2))
$FV(T') \cup (FV(T) \setminus \{x\})$.
Above, we also applied point (2) of the current proposition.
(4) By Definition 2.1.(1), we have:

$$T[y/x][z/y] = \mathit{Subst}\,(\mathit{Subst}\,(T, [y/x]), [z/y]) = \mathit{Subst}\,(T, [y/x]; [z/y]).$$

Now, for each $u \in \mathit{Var}$, we have that:

$$([y/x]; [z/y])(u) = \mathit{Subst}\,([y/x](u), [z/y]) = \left\{ \begin{array}{ll} \mathit{Subst}\,(y, [z/y]) & \text{, if } u = x \\ \mathit{Subst}\,(u, [z/y]) & \text{, if } u \neq x \end{array} \right. =$$

$$= \text{(by point (2))} = \left\{ \begin{array}{ll} z, & \text{if } u = x \\ z, & \text{if } u \neq x \text{ and } u = y \\ u, & \text{if } u \neq x \text{ and } u \neq y \end{array} \right. = \left\{ \begin{array}{ll} z, & \text{if } u = x \text{ or } u = y \\ u, & \text{if } u \neq x \text{ and } u \neq y \end{array} \right.$$

Hence, since $y \notin FV(T)$, it follows that $[y/x][z/y]\restriction_{FV(T)} = [z/x]\restriction_{FV(T)}$, implying, by Definition 2.1.(3), $\mathit{Subst}\,(T, [y/x]; [z/y]) = \mathit{Subst}\,(T, [z/x])$.
(5) Follows from point (4), since, by (2), $\mathit{Subst}\,(T, [x/x]) = \mathit{Subst}\,(T, \iota) = T$. □

Proof of Lemma 2.7. All points, except for (12), can be proved by induction either on the structure of formulas, or on the definition of $\equiv_\alpha$. We shall only prove the more interesting points, namely (1), (2), (6), (10) and (12) and refer the reader to [40] for full proofs of all points. (IH) will denote the induction hypothesis. Each time, we shall skip the starightforward inductive case of the logical connectives $\wedge, \Rightarrow$.

Points (1) and (2) are proved by induction on the structure of φ.
(1): Base case. $\rho\restriction_{FV(\pi(T_1,\ldots,T_n))} = \rho'\restriction_{FV(\pi(T_1,\ldots,T_n))}$ implies
$\rho\restriction_{T_i} = \rho'\restriction_{T_i}$ for each $i \in \{1, \ldots, n\}$, which implies (by Lemma 2.5)
$A_{T_i}(\rho) = A_{T_i}(\rho')$ for each $i \in \{1, \ldots, n\}$, which implies
$\rho \in A_\varphi$ iff $\rho' \in A_\varphi$.

Induction step. Assume $\rho\restriction_{FV(\forall x.\varphi)} = \rho'\restriction_{FV(\forall x.\varphi)}$. Then
$\rho\restriction_{FV(\varphi)\setminus\{x\}} = \rho'\restriction_{FV(\varphi)\setminus\{x\}}$, hence for all $a \in A$, $\rho[x \leftarrow a]\restriction_{FV(\varphi)} = \rho'[x \leftarrow a]\restriction_{FV(\varphi)}$.
By (IH), we obtain that for all $a \in A$, $\rho[x \leftarrow a] \in A_\varphi$ iff $\rho'[x \leftarrow a] \in A_\varphi$, in particular that $\rho \in A_{\forall x.\varphi}$ iff $\rho' \in A_{\forall x.\varphi}$.
(2): Base case. We have the following equivalences:
$\rho \in A_{\mathit{Subst}\,(\pi(T_1, \ldots, T_n), \theta)}$ iff
$\rho \in A_{\pi(\mathit{Subst}\,(T_1, \theta), \ldots, \mathit{Subst}\,(T_n, \theta))}$ iff
$(A_{\mathit{Subst}\,(T_1, \theta)}(\rho), \ldots, A_{\mathit{Subst}\,(T_n, \theta)}(\rho)) \in A_\pi$ iff (by Definition 2.4.(b.iii))
$(A_{T_1}(A_\theta(\rho)), \ldots, A_{T_n}(A_\theta(\rho))) \in A_\pi$ iff
$A_\theta(\rho) \in A_{\pi(T_1,\ldots,T_n)}$.

Induction step. We have the following equivalences: $\rho \in Subst\,(\forall x.\varphi, \theta)$ iff $\rho \in A_{\forall z.Subst\,(\varphi,\, \theta[x \leftarrow z])}$ (where z is the least variable of sort $sort\,(x)$ not in $FV(\varphi) \cup \bigcup\{\theta(y) : y \in FV(\varphi)\}$) iff
$\rho[z \leftarrow a] \in A_{Subst\,(\varphi,\, \theta[x \leftarrow z])}$ for all $a \in A$, iff (by (IH))
$A_{\theta[x \leftarrow z]}(\rho[z \leftarrow a]) \in A_\varphi$ for all $a \in A$, iff (as will be proved shortly **)
$A_\theta(\rho)[x \leftarrow a] \in A_\varphi$ for all $a \in A$, iff
$A_\theta(\rho) \in A_{\forall x.\varphi}$.
It remains to prove the equivalence **. It suffices to show that
$A_{\theta[x \leftarrow z]}(\rho[z \leftarrow a])\restriction_{FV(\varphi)} = A_\theta(\rho)[x \leftarrow a]\restriction_{FV(\varphi)}$.
Let $y \in FV(\varphi)$. Then

$$A_{\theta[x \leftarrow z]}(\rho[x \leftarrow a])(y) = A_{\theta[x \leftarrow z](y)}(\rho[x \leftarrow a]) =$$

$$= \begin{cases} A_{\theta(y)}(\rho[z \leftarrow a]), & \text{if } x \neq y \\ A_z(\rho[z \leftarrow a]), & \text{if } x = y \end{cases} = \begin{cases} A_{\theta(y)}(\rho[z \leftarrow a]), & \text{if } x \neq y \\ a, & \text{if } x = y. \end{cases}$$

On the other hand,

$$A_\theta(\rho)[x \leftarrow a](y) = \begin{cases} A_{\theta(y)}(\rho), & \text{if } x \neq y \\ a, & \text{if } x = y. \end{cases}$$

Finally, we need to show that $\rho[z \leftarrow a]\restriction_{FV(\theta(y))} = \rho\restriction_{FV(\theta(y))}$—this is true because, by the choice of z, $z \notin FV(\theta(y))$.

Points (3)-(11) can be proved by induction on the definition of $\equiv_\alpha$, and for (5)–(11) it is most convenient to use *mutual induction*. We only show how to handle points (6), (10) and (11).

Base case.
(6): We have that $Subst\,(\pi(T_1, \ldots, T_n), \iota) = \pi(Subst\,(T_1, \iota), \ldots, Subst\,(T_n, \iota)) = \pi(T_1, \ldots, T_n)$, hence $Subst\,(\pi(T_1, \ldots, T_n, \iota) \equiv_\alpha \pi(T_1, \ldots, T_n)$.
(10) and (11): Follow similarly to (6), using Definition 2.1.(3,6).

Inductive step.
(6): We need to prove that $Subst\,(\forall x.\varphi, \iota) \equiv_\alpha \forall x.\varphi$, i.e., that $\forall z.\varphi[z/x] \equiv_\alpha \forall x.\varphi$ with z as in the definition of substitution. Let z' such that $sort\,(z') = sort\,(x)$ and $z' \notin FV(\varphi) \cup FV(\varphi[z/x])$, or equivalently, $z' \notin FV(\varphi) \cup \{z\}$. Then, by (IH) for point (7), $\varphi[z/x][z'/z] \equiv_\alpha \varphi[z'/x]$ and we are done.
(10): Assume $\theta\restriction_{FV(\forall x.\varphi)} = \theta'\restriction_{FV(\forall x.\varphi)}$. In order to prove that $Subst\,(\forall x.\varphi, \theta) \equiv_\alpha Subst\,(\forall x.\varphi, \theta')$, note first that the variable z in the definition of substitution is the same in the two cases, and we need to show

$$\forall z.Subst\,(\varphi, \theta[x \leftarrow z]) \equiv_\alpha \forall z.Subst\,(\varphi, \theta'[x \leftarrow z]),$$

i.e., by (IH) for point (11), that

$$Subst\,(\varphi, \theta[x \leftarrow z]; [z'/z]) \equiv_\alpha Subst\,(\varphi, \theta'[x \leftarrow z]; [z'/z]),$$

i.e., that $Subst\,(\varphi, \theta[x \leftarrow z']) \equiv_\alpha Subst\,(\varphi, \theta'[x \leftarrow z'])$.

The last is true by (IH) for point (10), since $\theta[x \leftarrow z']\restriction_{FV(\varphi)} = \theta'[x \leftarrow z']\restriction_{FV(\varphi)}$.
(11): In order to prove that $Subst\,(\forall x.\varphi, \theta; \theta') \equiv_\alpha Subst\,(Subst\,(\forall x.\varphi, \theta), \theta')$, let

z, z', z'' as in the definition of substitution (for each of the three involved substitutions). We need to show that

$$\forall z.Subst\,(\varphi, (\theta;\theta')[x \leftarrow z]) \equiv_\alpha \forall z''.Subst\,(Subst\,(\varphi[x \leftarrow z'], \theta), \theta'[z' \leftarrow z'']),$$

i.e., that

$$Subst\,(\varphi, (\theta;\theta')[x \leftarrow z])[z'''/z]] \equiv_\alpha Subst\,(Subst\,(\varphi[x \leftarrow z'], \theta), \theta'[z' \leftarrow z''])[z'''/z''],$$

where $z''' \notin FV(\varphi) \cup \{z, z', z''\}$. Indeed, using (IH) for point (11) and the freshness of z, z', z'', z'''), we have the following chain of α-equivalences and equalities:

$$\begin{array}{c} Subst\,(\varphi, (\theta;\theta')[x \leftarrow z])[z'''/z] \equiv_\alpha Subst\,(\varphi, (\theta;\theta')[x \leftarrow z]; [z'''/z]) \equiv_\alpha \\ Subst\,(\varphi, (\theta;\theta')[x \leftarrow z''']) = Subst\,(\varphi, \theta[x \leftarrow z']; \theta'[z' \leftarrow z''']) = \\ Subst\,(\varphi, \theta[x \leftarrow z']; \theta'[z' \leftarrow z'']; [z'''/z'']) \equiv_\alpha \\ Subst\,(\varphi, \theta[x \leftarrow z']; \theta'[z' \leftarrow z''])[z'''/z''], \end{array}$$

which by (IH) for point (5) yield the desired result.

Finally, we prove (12): Again, the cases of the logical connectives $\wedge$ and $\Rightarrow$ are straightforward. For the $\forall$-case, assume $\varphi \equiv_\alpha \varphi'$. Then, by point (9), $\varphi[z/x] \equiv_\alpha \varphi'[z/x]$ for any x and z, in particular $\forall x.\varphi \equiv_\alpha \forall x.\psi$. □

C. Proof of completeness for TGL with equality (Th. 3.1$^=$)

In order to reduce TGL *with equality* to plain TGL, we consider the following set *Eql* of sentences called the *equality axioms*:

$x = x$	(Refl)
$x = y \Rightarrow y = x$	(Symm)
$x = y \wedge y = z \Rightarrow x = z$	(Trans)
$(x_1 = y_1 \wedge \ldots \wedge x_n = y_n \ \wedge \pi(x_1, \ldots, x_n)) \Rightarrow \pi(y_1, \ldots, y_n)$	(Comp$_\pi$)
$x = y \Rightarrow T[x/z] = T[y/z]$	(Subst)

Soundness: The soundness of all the rules except for (Sbs) is straightforward.

Let $\mathcal{A}$ be a model and ρ a sort-preseving interpretation such that $A_{T_1}(\rho) = A_{T_2}(\rho)$. Then, by Definition 2.4.(b.iii), we obtain
$A_{T[T_1/x]}(\rho) = A_T(\rho[x \leftarrow A_{T_1}(\rho)]) = A_T(\rho[x \leftarrow A_{T_2}(\rho)]) = A_{T[T_2/x]}(\rho)$.
This immediately implies the soundness of (Sbs).

Completeness: Adding the equality rules amounts to adding the equality axioms *Eql* (listed in Section 2.4) in the antecedent of sequents. Thus $\Gamma \triangleright \Delta$ is provable in $\mathcal{G}_=$ iff $Eql \cup \Gamma \triangleright \Delta$ is provable in $\mathcal{G}$.

Let $\mathcal{L}$ be the given TGL language with equality. We define $\mathcal{L}'$ to be the language without equality that has the term syntax and the ordinary relation symbols of $\mathcal{L}$, and additionally includes an ordinary binary relation symbol $=_s \in \Pi'_{ss}$ for each sort s. For each $\mathcal{L}$-formula φ, let $\varphi^\#$ be the $\mathcal{L}'$-formula obtained by replacing each occurrence of $=$ with $=_s$ for the appropriate sort s. Let $Eql' = \{\varphi^\#.\ \varphi \in Eql\}$.

For every $\mathcal{L}'$-model $\mathcal{A}'$ satisfying Eql', we construct an $\mathcal{L}$-model by a straightforward quotienting process: We define the relation $\equiv$ by $a \equiv b$ iff $Asort'(a) = Asort'(b)$ and $A' \models_\rho x =_{Asort(a)} y$ for some ρ with $\rho(x) = a$ and $\rho(y) = b$. Due to satisfaction of Eql' by $\mathcal{A}'$, $\equiv$ is an equivalence compatible with sorting, with the relations A'_π and with substitution, the last in the sense that $A'_{T_1}(\rho) = A'_{T_2}(\rho)$ implies $A'_{T[T_1/x]}(\rho) = A'_{T[T_2/x]}(\rho)$. Thus, we can define a quotient model $\mathcal{A}$ as follows:
- $A = A'/_{\equiv}$;
- $Asort(a) = Asort'(a')$ for some a' in (the $\equiv$-class) a;
- $A_T(\rho) = A'_T(\rho')$, where, for each x, $\rho'(x)$ is defined as some element in $\rho(x)$;
- $A_\pi(a_1, \ldots, a_n) = A'_\pi(a'_1, \ldots, a'_n)$ where each a'_i is some element in a_i.

For each sort-preserving $\rho' : Var \to A'$, we define $\rho'^{\#} : Var \to A$ by $\rho'^{\#}(x) = \rho'(x)/_{\equiv}$. Then ρ' is also sort-preserving and an easy induction on φ shows that $\mathcal{A}' \models_{\rho'} \varphi^{\#}$ iff $A \models_{\rho'^{\#}} \varphi$. It follows that $\Gamma \triangleright \Delta$ is tautological in $\mathcal{L}$ in the logic with equality iff $(Eql', \Gamma) \triangleright \Delta$ is tautological in $\mathcal{L}'$ in the logic without equality. Completeness of $\mathcal{G}_=$ now follows from completeness of $\mathcal{G}$. □

D. Proofs on the comparison between ad hoc and TGL models

D.1. For the untyped λ-calculus.

Proof of Lemma 5.1. The inverse of the forgetful function maps simple pre-structures $(A, _\langle_\rangle, (A_X)_{X \in Term})$ verifying (P1)-(P4) to $(A, _\langle_\rangle, (A_X)_{X \in Term(A)})$, where for each term X in $Term$, sequence of elements $a_1, \ldots, a_n$ in A and sequence of distinct variables $x_1, \ldots, x_n$, $A_{X[a_1/x_1, \ldots, a_n/x_n]}(\rho)$ is, by definition, $A_X(\rho[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n])$. This definition is correct, because any term in $Term(A)$ has the form $X[a_1/x_1, \ldots, a_n/x_n]$ for some X, and $A_X(\rho[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n])$ does not depend on the choice of X. A simple induction on the structure of $Term(A)$ terms shows that $A_{X[a_1/x_1, \ldots, a_n/x_n]}(\rho)$ is indeed equal to $A_X(\rho[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n])$ in any pre-structure, and thus the two mappings are mutually inverse.

These mappings obviously preserve satisfaction. As for properties (P5) and (P6), one needs another induction to prove that in a pre-structure, verifying these properties w.r.t. $Term$-terms is sufficient for them to hold w.r.t. $Term(A)$-terms; here one uses again the equality $A_{X[a_1/x_1, \ldots, a_n/x_n]}(\rho) = A_X(\rho[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n])$. □

Proof of Lemma 5.2. Since (β'), (η') and (ext) are instances of the schemas (β), (η) and (ext), all we need to show is that the latter follow from the former; and this simply holds because in our logic it is sound to infer $\varphi(X)$ from $\forall y.\varphi(y)$, and by the way substitution in formulas was defined. □

Proof of Proposition 5.3. We show that $\mathcal{A}^{\#}$ is a TGL model. The TGL model axiom (b).(i) holds trivially, since the syntax is single-sorted. Moreover, (b).(ii) is precisely (P1).

The remaining axiom, (b).(iii), requires checking that $A_{Subst(X,\theta)}(\rho) = A_X(A_\theta(\rho))$ for each ρ and θ. We first reduce the problem to finite substitutions,

i.e., substitutions θ' such that the support of θ', namely $\{x \in Var \mid \theta'(x) \neq x\}$, is finite. Let θ' be the finite substitution defined by $\theta'(x) = \theta(x)$ if $x \in FV(X)$ and $= x$ otherwise. Form the definition of θ' and Definition 2.1.(3), we have $Subst\,(X, \theta) = Subst\,(X, \theta')$, and hence $A_{Subst\,(X,\theta)}(\rho) = A_{Subst\,(X,\theta')}(\rho)$; moreover, from the definition of θ' we have $A_\theta(\rho)\restriction_{FV(X)} = A_{\theta'}(\rho)\restriction_{FV(X)}$, and hence, by (P3), $A_X(A_\theta(\rho)) = A_X(A_{\theta'}(\rho))$.

Now, for any finite substitution θ', if $x_1, \ldots, x_n$ are all the distinct variables of its support and $X_i = \theta(x_i)$, we have $Subst\,(X, \theta') = X[X_1/x_1, \ldots, X_n/x_n]$ and $A_{\theta'}(\rho) = \rho[x_1 \leftarrow A_{X_1}(x_1), \ldots, x_n \leftarrow A_{X_n}(x_n)]$. It therefore suffices to prove $A_{X[X_1/x_1,\ldots,X_n/x_n]}(\rho) = A_X(\rho[x_1 \leftarrow A_{X_1}(x_1), \ldots, x_n \leftarrow A_{X_n}(x_n)])$. We do this by lexicographic induction on two criteria: the depth of X, and then the number n. The cases with X variable and X of the form $X'\,X''$ are easy, and they use (P1) and (P2). Assume now that X has the form $\lambda x.X'$. Since we work modulo α-equivalence, we can assume that x is not free in any of $X_1, \ldots, X_n$.

- If $x \notin \{x_1, \ldots, x_n\}$, then $X[\overline{X}/\overline{x}] = \lambda x.(X'[\overline{X}/\overline{x}])$. Thus we need to check $A_{\lambda x.(X'[\overline{X}/\overline{x}])}(\rho) = A_{\lambda x.X'}(\rho[\overline{x} \leftarrow A_{\overline{X}}(\rho)])$. By (P4), it is sufficient to consider $a \in A$ and prove $A_{X'[\overline{X}/\overline{x}]}(\rho[x \leftarrow a]) = A_{\lambda x.X'}(\rho[\overline{x} \leftarrow A_{\overline{X}}(\rho), x \leftarrow a])$, i.e., $A_{X'[\overline{X}/\overline{x}]}(\rho[x \leftarrow a]) = A_{\lambda x.X'}(\rho[x \leftarrow a][\overline{x} \leftarrow A_{\overline{X}}(\rho)])$, which is true by the induction hypothesis applied to X' and $\rho[x \leftarrow a]$.

- If $x \in \{x_1, \ldots, x_n\}$, say $x = x_1$, then $X[\overline{X}/\overline{x}] = X[X_2/x_2, \ldots, X_n/x_n]$ and by the induction hypothesis (second criterion), $X[X_2/x_2, \ldots, X_n/x_n] = A_X(\rho[x_2 \leftarrow A_{X_2}(x_1), \ldots, x_n \leftarrow A_{X_n}(x_n)]$. Finally, because $x \notin FV(X)$, the valuations $\rho_1 = \rho[\overline{x} \leftarrow A_{\overline{X}}(\rho)]$ and $\rho_2 = \rho[x_2 \leftarrow A_{X_2}(x_1), \ldots, x_n \leftarrow A_{X_n}(x_n)]$ coincide on $FV(X)$, hence, by (P3), $A_X(\rho_1) = A_X(\rho_2)$ and we are done.

(Above, we used the obvious tuple notations $\overline{x}$ for $(x_1, \ldots, x_n)$, $\overline{X}$ for $(X_1, \ldots, X_n)$ etc.) $\mathcal{A}^\#$ satisfies (ξ) because (P4) is just a semantic statement of (ξ).

We show that $\mathcal{A}^\$$ is a λ-model. Properties (P1) and (P3) are required for generic models as well (one as an axiom and the other as a proved fact), hence they hold. (P2) holds as an instance of the axiom (b).(iii) of TGL models: $A_{X_1\,X_2}(\rho) = A_{x_1x_2[X_1/x_1,X_2/x_2]}(\rho) = A_{x_1x_2}([\rho[x_1 \leftarrow A_{X_1}(\rho), x_2 \leftarrow A_{X_2}(\rho)]) = A_{X_1}(\rho)\langle A_{X_2}(\rho)\rangle$. (We also used the definition of $_\langle_\rangle$.) Again, (P4) holds in $\mathcal{A}^\$$ because M satisfies (ξ).

That # and \$ are mutually inverse follows from the fact that $a\langle b\rangle = A_{xy}(\rho)$, with ρ mapping x to a and y to b, holds in any pre-structure verifying (P1)-(P4).

In order to see that the pre-structure is a λ-model (i.e., it also verifies (P5) iff the corresponding TGL model satisfies (β), note that (P5) is just a semantic statement of (β'), which is equivalent to (β) by Lemma 5.2. Similarly, extensional λ-models correspond to $(\beta) \cup (\eta)$-TGL models because of the following:
- under the $(\beta), (\xi)$ assumptions, (η) is equivalent to (ext);
- (ext) is equivalent to (ext') by Lemma 5.2;
- (ext') is simply a semantic statement of (P6).

Finally, both # and \$ preserve satisfaction since it has the same definition for equations in pre-structures and generic models. □

D.2. For System F.

Lemma 1. *We consider the context of the "Henkin to TGL" definitions. The following hold:*
(1) The definition of M_X does not depend on the choice of Γ and T.
(2) $\mathcal{M}$ is a TGL model.
(3) $\mathcal{M} \models_{TGL} \mathcal{SF}_{=}$.

Proof. (1): Assume $\vdash_{TSF} \Gamma \triangleright X : T$, $\vdash_{TSF} \Gamma' \triangleright X : T'$ and (γ, δ) is compatible with Γ and Γ'. Then $H_T(\gamma) = H_{T'}(\gamma)$ and $H_{\Gamma \triangleright X:T}(\gamma, \delta) = H_{\Gamma' \triangleright X:T'}(\gamma, \delta)$ follow by induction on the derivation of $\vdash_{TSF} \Gamma \triangleright X : T$, employing inversion rules for $\vdash_{TSF} \Gamma' \triangleright X : T'$ and conditions (4)-(10) from Definition 5.6.
(2): We verify condition b.(i)–b.(iii) from the definition of TGL models. b.(i) is immediate. b.(ii) amounts to checking $M_t(\gamma) = \gamma(t)$, which follows from Definition 5.6.(3), and $M_x(\gamma, \delta) = \delta(x)$. To check the latter, we let $\Gamma = x : \gamma(t)$ for some t. Then $\vdash_{TSF} \Gamma \triangleright x : t$ and (γ, δ) is compatible with Γ—hence, by point (1) and Definition 5.6.(6), we have $M_x(\gamma, \delta) = H_{\Gamma \triangleright x:t}(\gamma, \delta) = \delta(x)$, as desired. It remains to verify condition b.(iii), which amounts to checking that $M_{Subst\,(T,\omega)}(\gamma) = M_T(M_\omega(\gamma))$ and $M_{Subst\,(X,(\omega,\theta))}(\gamma, \delta) = M_X(M_{(\omega,\theta)}(\gamma, \delta))$. These follow from the following properties of the Henkin interpretations:
- $H_{Subst\,(T,\omega)}(\gamma) = M_T(H_\omega(\gamma))$. This is provable by induction on T.
- If $\vdash_{TSF} \Gamma \triangleright X : T$, $\vdash_{TSF} \Gamma_x \triangleright \delta(x) : T_x$ for all $x \in FV_{data}(X)$, and (γ, δ) is compatible with Γ and the Γ_x's, then there exists Γ' such that
$\vdash_{TSF} \Gamma' \triangleright Subst\,(X, (\omega, \theta)) : Subst\,(T, \omega)$ and (γ, δ) is compatible with Γ'. This is provable by induction on the derivation of $\vdash_{TSF} \Gamma \triangleright X : T$.
(3): The typing axioms—[Abs], [App], [T-Abs] and [T-App]—follow from the types of the Henkin operators, Definition 5.6.(d-g). The equational axioms follow similarly to the ones for the untyped λ-calculus in Subsection 5.1. □

Lemma 2. *We consider the context of the "TGL to Henkin" definitions. The following hold:*
(1) The definition of $\Pi(f)$ does not depend on the choice of t, T, and γ.
(2) The definition of $App_{\tau,\sigma}(d)(d')$ does not depend on the choice of γ and δ.
(3) The definition of $App_f(\tau)$ does not depend on the choice of γ.
(4) $\mathcal{H}$ is a Henkin model.

Proof. (1)-(3): Follow from $\mathcal{M} \models_{TGL} (\xi)$ together with the definition of model. For example, (1) follows like this: Let t, T, γ and t', T', γ' such that $f = (\tau \mapsto M_T(\gamma[t \leftarrow \tau])) = (\tau \mapsto M_{T'}(\gamma'[t' \leftarrow \tau]))$ (*). Using Definition 2.4.(b.iii) and Lemma 2.5, we may assume, perhaps after choosing some fresh variables and substituting them for the variables of T', that $t = t'$ and $FV(T) \cap FV(T') \subseteq t$. We define γ'' to behave like γ on $FV(T) \setminus \{t\}$ and like γ' on $FV(T') \setminus \{t\}$. By (*) and Lemma 2.5, we obtain $(\tau \mapsto M_T(\gamma''[t \leftarrow \tau])) = (\tau \mapsto M_{T'}(\gamma''[t \leftarrow \tau]))$. By $\mathcal{M} \models_{TGL} (\xi)$, we have $M_{\Pi t.T}(\gamma'') = M_{\Pi t.T'}(\gamma'')$, hence, by Lemma 2.5 again, we have $M_{\Pi t.T}(\gamma) = M_{\Pi t.T'}(\gamma')$, as desired.
(4): Follows from points (1)-(3), the axioms of $\mathcal{SF}_{=}$, and Proposition 5.5, the last taking care of the axioms' typing assumptions. The proofs are again

similar to the ones for the untyped λ-calculus. For example, let us prove condition (9) from Definition 5.6. We assume $\vdash_{TSF}\ tj$, where tj is $\Gamma \rhd \lambda x : T.\, X\ :\ T \to T'$, and (γ, δ) is compatible with Γ. Let φ, ψ, and χ be the TGL formulas $\bigwedge \Gamma^{\#}$, $tpOf(\lambda x : T.\, X, T \to T')$, and $\varphi \to \psi$, respectively. By Proposition 5.5, we have $\mathcal{SF} \vdash_{TGL} \chi$, hence $\mathcal{M} \models_{TGL} \chi$, hence $(\gamma, \delta) \in M_\chi$. Moreover, by the Γ-compatibility of (γ, δ), we have $(\gamma, \delta) \in M_\varphi$. Hence $(\gamma, \delta) \in M_\psi$, that is, $(M_{\lambda x:T.\, X}(\gamma, \delta), M_\to(M_T(\gamma, \delta), M_{T'}(\gamma, \delta))) \in M_{tpOf}$, that is, $H_{tj} \in \mathcal{D}om_{H_T(\gamma) \to H'_T(\gamma)}$, which proves the first part of (9).

To prove the second part, let $d \in \mathcal{D}om_{H_T(\gamma)}$ and $\delta' = \delta[x \leftarrow d]$. Then
$App_{H_T(\gamma), H_{T'}(\gamma)}(H_{\Gamma \rhd \lambda x:T.X\,:\,T \to T'}(\gamma, \delta))(d) =$
$M_{__}(M_{\lambda x:T.\, X}(\gamma, \delta),\ M_x(\gamma, \delta')) =$ (by Lemma 2.5)
$M_{__}(M_{\lambda x:T.\, X}(\gamma, \delta'),\ M_x(\gamma, \delta')) =$
$M_{(\lambda x:T.\, X)\, x}(\gamma, \delta') =$ (since $\mathcal{M} \models_{TGL} (\beta)$ and $\mathcal{M} \models_{TGL} \chi$)
$M_X =$ (since (γ, δ') is compatible with $\Gamma, x\!:\!T$)
$H_{\Gamma, x:T \rhd X:T'}(\gamma, \delta')$, as desired. □

Proof of Proposition 5.7. Let tj be $\Gamma \rhd X\!:\!T$ and tj' be $\Gamma \rhd Y\!:\!T$. Let φ be the TGL formula $\bigwedge \Gamma^{\#}$. At point (1), we let $\mathcal{M} = \mathcal{H}^{\#}$. At point (2), we let $\mathcal{H} = \mathcal{M}^{\$}$. In both cases, immediately from the definitions, we have:
(γ, δ) compatible with Γ in $\mathcal{H}$ iff $(\gamma, \delta) \in M_\varphi$ (*)

We prove that, in both cases, assuming one of the equivalent conditions from (*) holds, we have: $M_X(\gamma, \delta) = H_{tj}(\gamma, \delta)$ (**)

Indeed, at point (1) this follows from Lemma 1, and at point (1) it follows directly from the definition of $\mathcal{M}^{\$}$.

(*) and (**) immediately imply the desired facts. □